

Final Report: Introduction to Devops Terraform Docker Kubernetes

Introduction

This project showcases the development of a sophisticated real-time e-commerce analytics system using AWS services. Leveraging the power of Amazon SQS for asynchronous message queuing and DynamoDB for scalable data storage, this solution enables businesses to capture, process, and analyze e-commerce data in real-time.

Through this analytics system, e-commerce businesses can monitor key metrics such as website traffic, product sales, and inventory levels in real-time. By gaining instant insights into customer preferences, market trends, and inventory status, businesses can make informed decisions to optimize marketing strategies, streamline operations, and enhance the overall customer experience.

Technologies Used in this project:

1. Amazon Web Services (AWS):

- **Amazon SQS (Simple Queue Service):** Asynchronous message queuing service used to decouple the components of the application. Messages are sent to SQS queues by producers and retrieved by consumers, enabling reliable communication between different parts of the system.
- **DynamoDB:** Fully managed NoSQL database service provided by AWS. DynamoDB offers seamless scalability, high performance, and low latency for storing and retrieving data. In this project, DynamoDB is used for persisting data received from the SQS queue.

- **AWS Lambda:** Serverless compute service used to execute code in response to events. In this project, Lambda functions can be triggered by messages arriving in the SQS queue, allowing for seamless data processing and integration with other AWS services.
- **IAM Roles (Identity and Access Management):** IAM roles are used to define permissions and access control policies for AWS resources. IAM roles ensure that the Lambda functions have the necessary permissions to interact with other AWS services, such as SQS and DynamoDB, while following the principle of least privilege.

2. Python:

- Python programming language is used for writing the backend logic and interacting with AWS services. Python's boto3 library is particularly useful for programmatically accessing and managing AWS resources.

3. JSON (JavaScript Object Notation):

- Lightweight data interchange format used for serializing and transmitting data between components of the application.

4. IDE (Integrated Development Environment):

- Any IDE or text editor can be used for writing and executing Python code, such as VS Code, PyCharm, or Jupyter Notebook.

5. AWS Access Key and Secret Access Key:

- Authentication credentials required for accessing AWS services programmatically.

6. boto3 Library:

- Python SDK for AWS, used for interacting with AWS services from Python code.

7. Web Browser:

- To access and interact with the Streamlit web application or any other web-based interfaces for managing AWS services.

Task 1: Launch EC2 Instance and install Docker

Provisioning an Ubuntu EC2 Instance with Docker Installed using AWS CloudFormation

Introduction:

In response to the requirement of TELEMAX, an airline management company, to deploy an effective architecture for storing real-time data in a NoSQL-based database on AWS cloud, this report outlines the steps to create an Ubuntu EC2 instance with Docker installed using AWS CloudFormation.

Step 1: Creating an AMI Template

- Launch an EC2 instance from an existing AMI (Amazon Machine Image) or create a new instance with the desired configuration.
- Customize the instance according to your requirements by installing software, configuring settings, and making any necessary changes.
- Create an AMI from the instance using the AWS Management Console, CLI, or SDKs. This AMI will serve as the template for creating new instances with the same configuration and setup.

Amazon Machine Images (AMIs) (1) [Info](#)

Name	AMI name	AMI ID	Source	Own
<input type="checkbox"/> Ubuntu-ami	Ubuntu-ami	ami-07505ab8962e2c5c2	905418082383/Ubuntu-ami	9054

Select an AMI

EC2 > AMIs > ami-07505ab8962e2c5c2

Image summary for ami-07505ab8962e2c5c2

AMI ID	Image type	Platform details	Root device type
ami-07505ab8962e2c5c2	machine	Linux/UNIX	EBS

AMI name	Owner account ID	Architecture	Usage operation
Ubuntu-ami	905418082383	x86_64	RunInstances

Root device name	Status	Source	Virtualization type
/dev/sda1	Available	905418082383/Ubuntu-ami	hvm

Boot mode	State reason	Creation date	Kernel ID
-	-	Mon Mar 04 2024 09:53:08 GMT+0530 (India Standard Time)	-

Description	Product codes	RAM disk ID	Deprecation time
Ubuntu 20.04 LTS	-	-	-

Last launched time	Block devices
Mon Mar 04 2024 14:30:45 GMT+0530 (India Standard Time)	/dev/sda1=snap-04a15f99465ea0588:8:true:cpn2

Step 2: Creating a CloudFormation Template

- Define the resources required for your infrastructure in a CloudFormation template. This includes EC2 instances, security groups, IAM roles, etc.
- Specify the properties and configurations for each resource, such as instance type, key pair, security group rules, etc.
- Deploy the CloudFormation stack using the AWS Management Console, CLI, or SDKs, which will create the specified resources based on the template.

Template

```
AWS::TemplateFormatVersion: '2010-09-09'
Description: Create an Ubuntu EC2 instance with Docker instal.

Parameters:
  InstanceType:
    Description: EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t2.medium
      - t3.micro
      - t3.small
      - t3.medium
    ConstraintDescription: Must be a valid EC2 instance type.

  KeyName:
    Description: Name of an existing EC2 key pair for SSH access
    Type: AWS::EC2::KeyPair::KeyName

Resources:
  EC2SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH access to the EC2 instance
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0 # Allow SSH access from anywhere

  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-07505ab8962e2c5c2 # Ubuntu 20.04 LTS (HVM
```

```

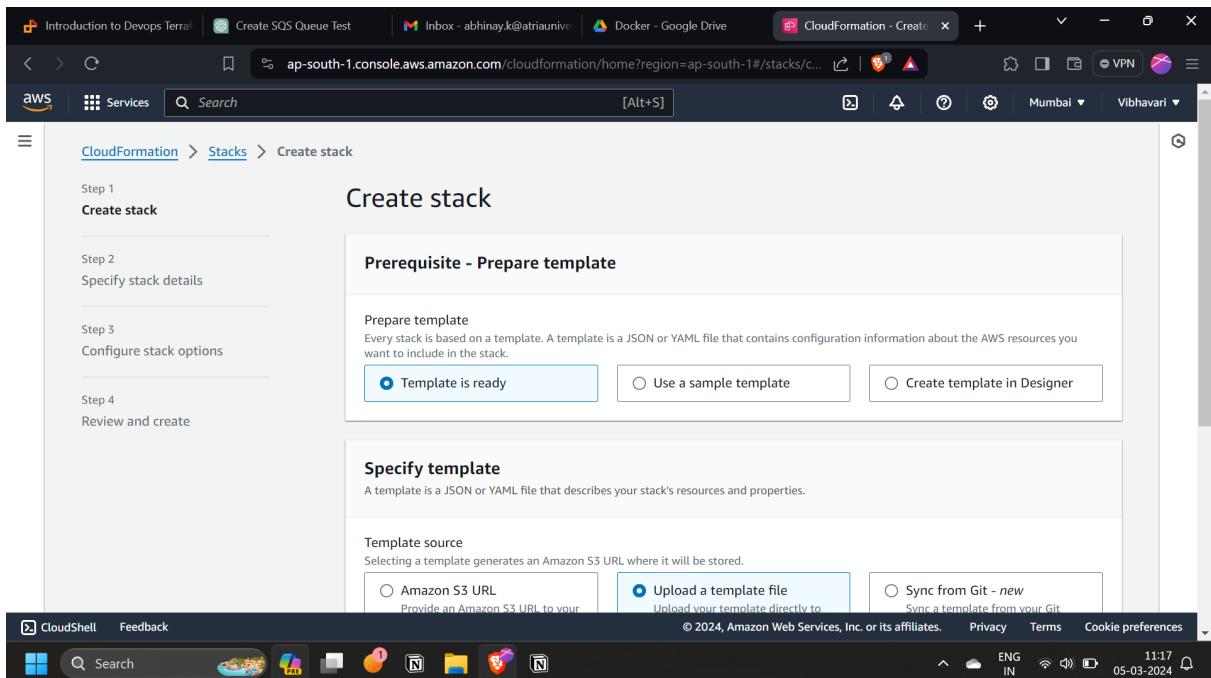
InstanceType: !Ref InstanceType
KeyName: ab-task1-key
SecurityGroups:
  - Ref: EC2SecurityGroup
UserData:
  Fn::Base64: |
    #!/bin/bash
    apt-get update -y
    apt-get install -y docker.io

Outputs:
InstanceId:
  Description: Instance ID of the newly created EC2 instance
  Value: !Ref EC2InstanceId
Export:
  Name: EC2InstanceId

```

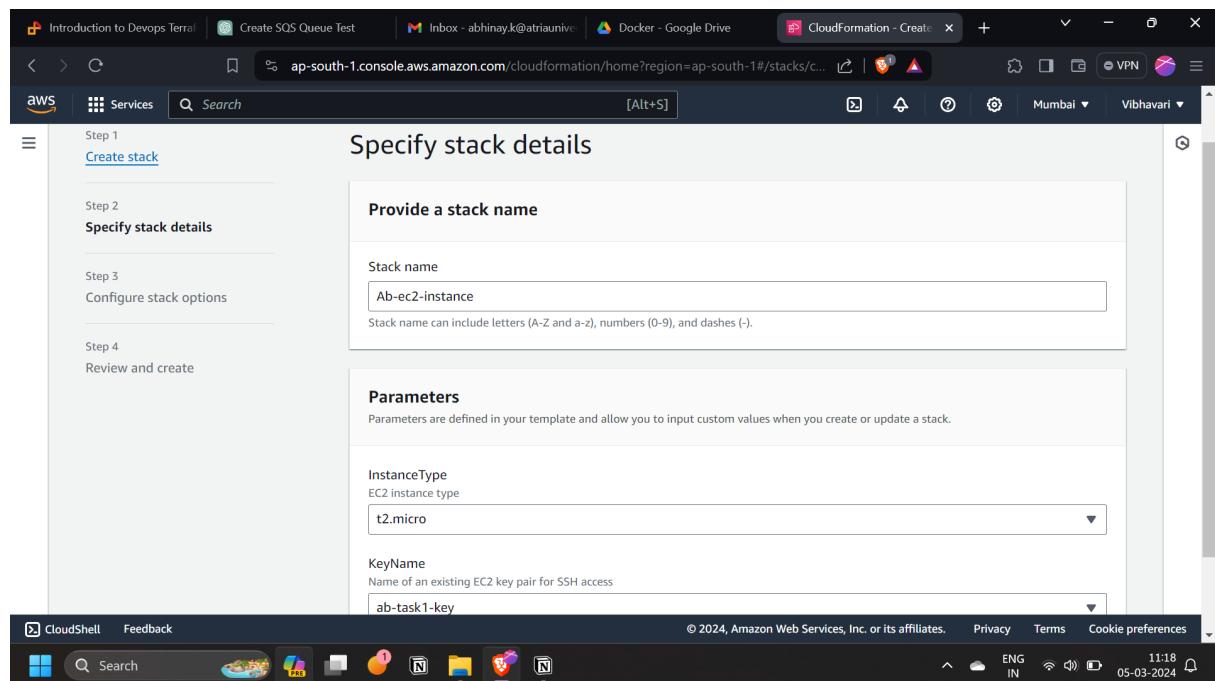
Step 3: Uploading CloudFormation Template:

Click on the "Create stack" button and choose the option "With new resources (standard)". Select "Upload a template file" and upload the CloudFormation template file provided.



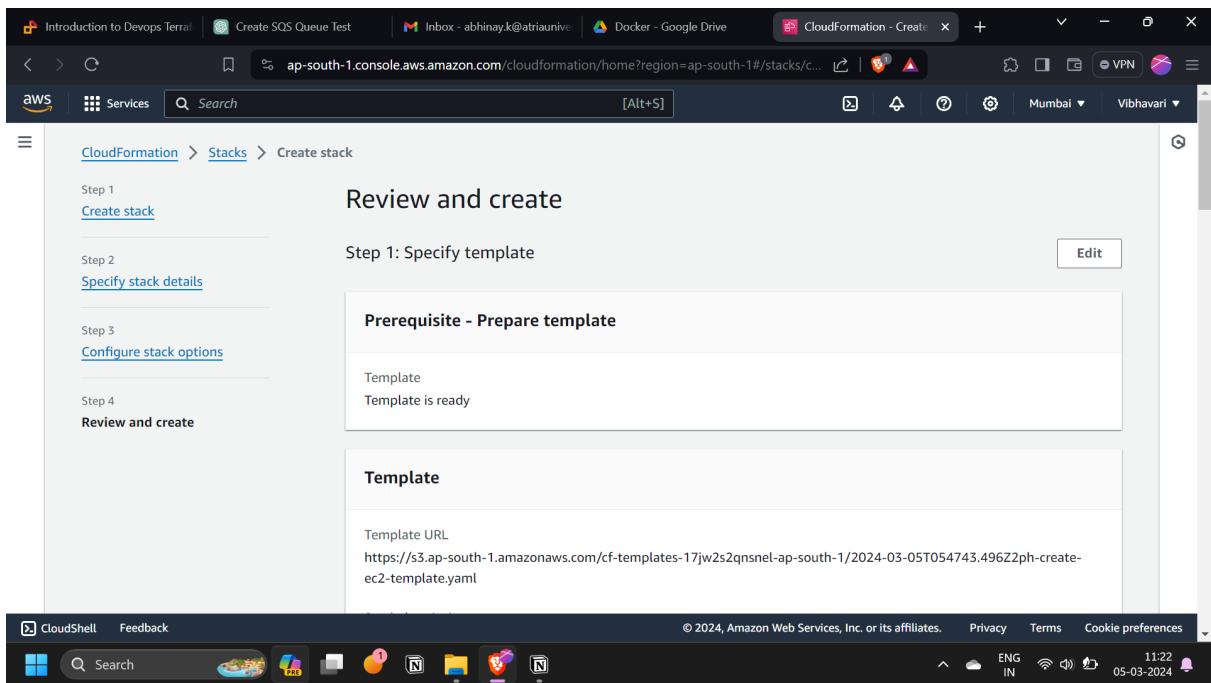
Step 4: Configuring Stack Parameters:

Enter a stack name and specify any required parameters such as the key pair name. Proceed to the next steps, confirming the default settings or making adjustments as necessary.



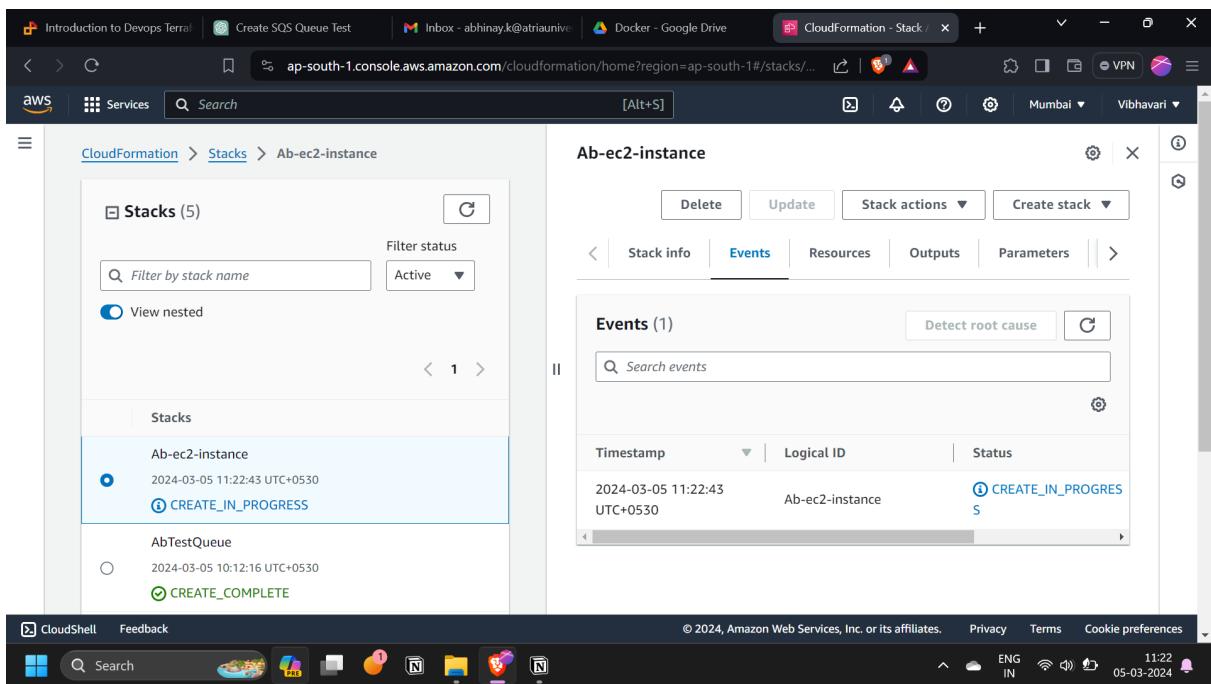
Step 5: Reviewing and Creating the Stack:

Review the stack configuration and click on "Create stack" to initiate the provisioning process.



Step 6: Monitoring Stack Creation:

Navigate to the "Stacks" section within CloudFormation and monitor the progress of stack creation. Wait until the status changes to "CREATE_COMPLETE".



The screenshot shows the AWS CloudFormation console with the 'Events' tab selected for the stack 'Ab-ec2-instance'. The table lists a single event:

Timestamp	Logical ID	Status
2024-03-05 11:22:43 UTC+0530	Ab-ec2-instance	CREATE_IN_PROGRESS

Step 7: Accessing EC2 Instance:

Once the stack creation is complete, navigate to the EC2 service from the AWS Management Console. Locate the newly created EC2 instance.

The screenshot shows the AWS CloudFormation console with the 'Resources' tab selected for the stack 'Ab-ec2-instance'. The table lists two resources:

Logical ID	Physical ID	Type	Status
EC2Instance	03acd069d07393ba6	AWS::EC2::Instance	CREATE_COMPLETE
EC2SecurityGroup	Ab-ec2-instance-EC2SecurityGroup-AYQK174DZ7FH	AWS::EC2::SecurityGroup	CREATE_COMPLETE

The screenshot shows the AWS EC2 Instances page. On the left, a sidebar menu includes 'EC2 Dashboard', 'EC2 Global View', 'Events', 'Instances' (selected), 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Capacity Reservations', and 'New'. Under 'Images', there are 'AMIs' and 'AMI Catalog'. The main content area displays 'Instances (1) Info' with a search bar and a table. The table has columns: Name, Instance ID, Instance state, Instance type, Status check, and Alarm state. One row is shown: 'Ab-ec2-instance' (Instance ID: i-03acd069d07393ba6), which is 'Running' (Status check: Initializing). Below the table is a modal titled 'Select an instance'.

Step 8: Verifying Docker Installation:

Connect to the EC2 instance using SSH and verify that Docker is installed properly by running the command

```
docker --version .
```

The screenshot shows the AWS CloudShell terminal. The command 'docker --version .' was run, resulting in the following output:

```
39 updates can be applied immediately.
25 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-32-47:~$
```

At the bottom, it shows the instance details: 'i-03acd069d07393ba6 (Ab-ec2-instance)' and 'Public IPs: 13.200.249.44 Private IPs: 172.31.32.47'.

```

Memory usage: 27%           IPv4 address for docker0: 172.17.0.1
Swap usage:  0%             IPv4 address for eth0:   172.31.32.47

Expanded Security Maintenance for Applications is not enabled.

39 updates can be applied immediately.
25 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-32-47:~$ docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
ubuntu@ip-172-31-32-47:~$ 

```

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 11:26 05-03-2024

Step 9: Testing Docker:

Run a simple Docker command, such as

`docker run hello-world`, to ensure that Docker is functioning correctly on the EC2 instance.

```

ubuntu@ip-172-31-45-231:~$ sudo -i
root@ip-172-31-45-231:~# docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
root@ip-172-31-45-231:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:000bc56993/abe195e20322a0bde6b2922d805332fd6d8a68b19f524b7d21d
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

```

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 14:37 04-03-2024

Conclusion

The Ubuntu EC2 instance with Docker installed has been successfully provisioned using AWS CloudFormation, providing a platform for further development and deployment of TELEMAX's architecture.

Task 2 : Create SQS Queue

Create a SQS Queue and Test with Dummy Data: Manually

1. Create a SQS Queue:

- Go to the AWS Management Console and navigate to the Amazon SQS service.
- Click on "Create Queue".
- Enter a name for your queue and configure any additional settings as needed (such as message retention period, visibility timeout, etc.).
- Click "Create Queue" to create the SQS queue.

2. Test with Dummy Data:

- After creating the SQS queue, navigate to the "Send and receive messages" section.
- Click on "Send a message".
- Enter some dummy data (e.g., a JSON message) and click "Send message".
- Verify that the message is successfully sent to the queue.

Create a SQS Queue and Test with Dummy Data with CloudFormation Template

Create a CloudFormation Template:

Template code

```
AWSTemplateFormatVersion: '2010-09-09'
```

Resources:

MyQueue:

Type: "AWS::SQS::Queue"

Properties:

QueueName: "AbTestQueue"

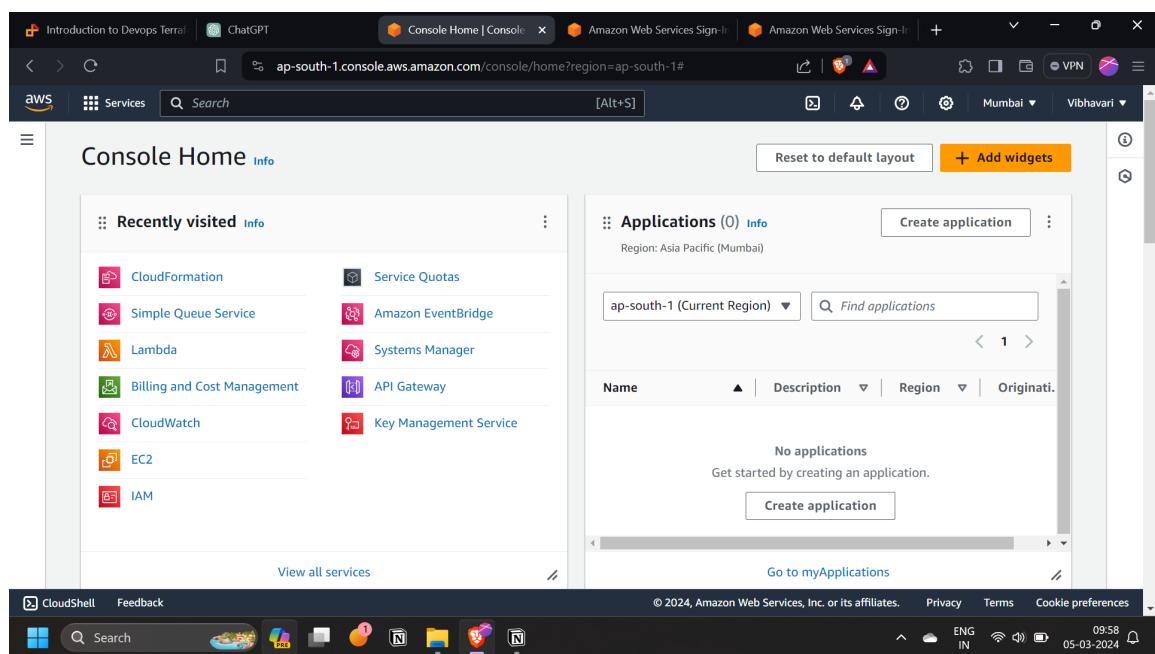
Outputs:

QueueURL:

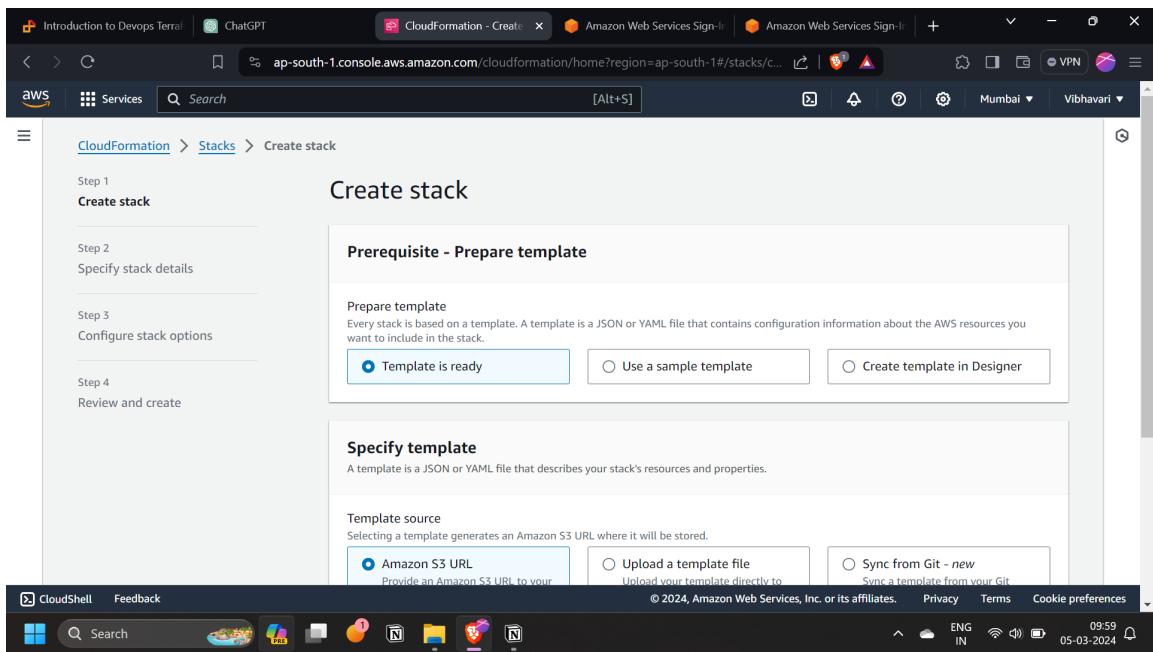
Description: "URL of the SQS queue"

Value: !GetAtt AbTestQueue.QueueUrl

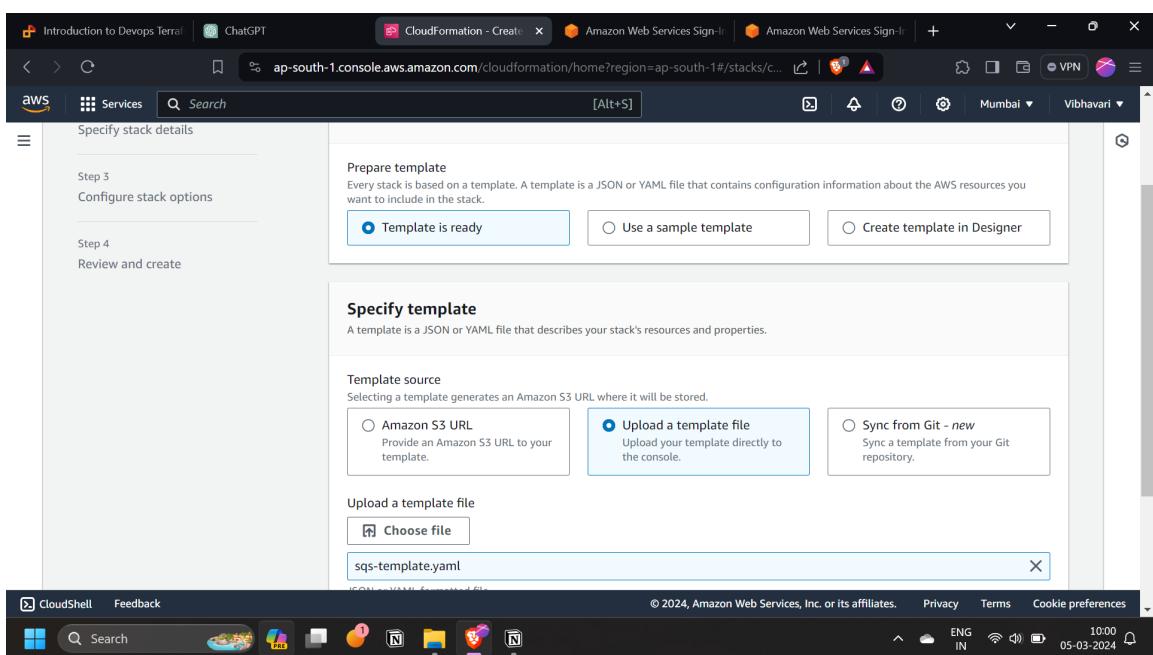
- Log in to the AWS Management Console and navigate to the CloudFormation service.



- Click on "Create Stack" to create a new stack.

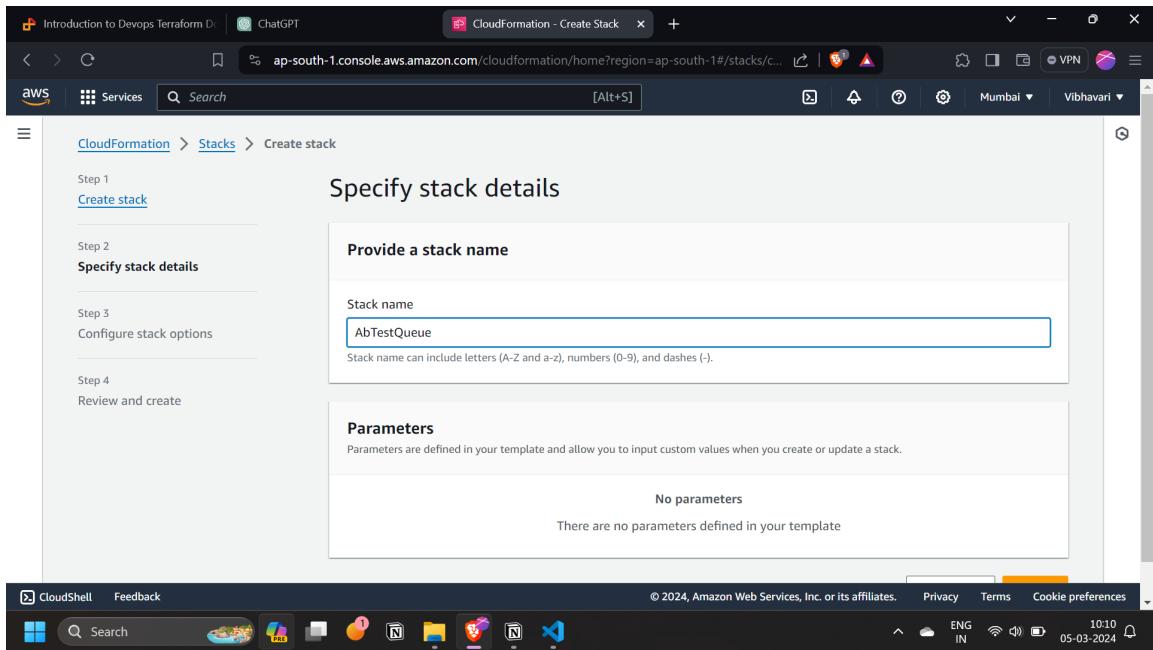


- Select "Template is ready" and "Upload a template file".

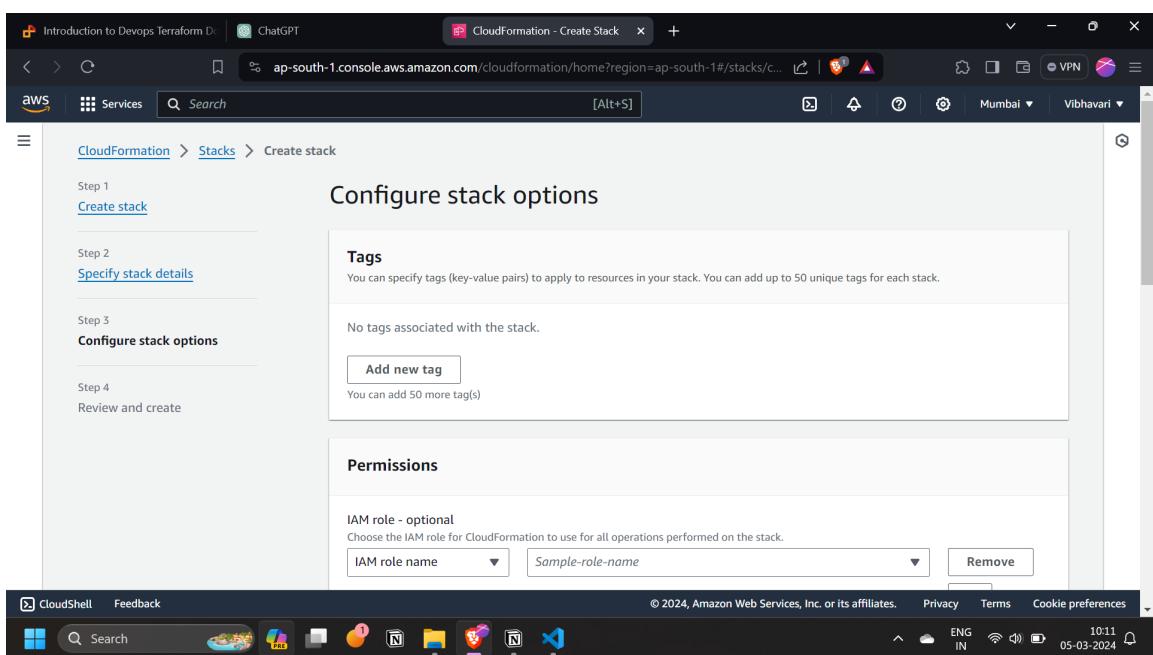


Deploy the CloudFormation Stack:

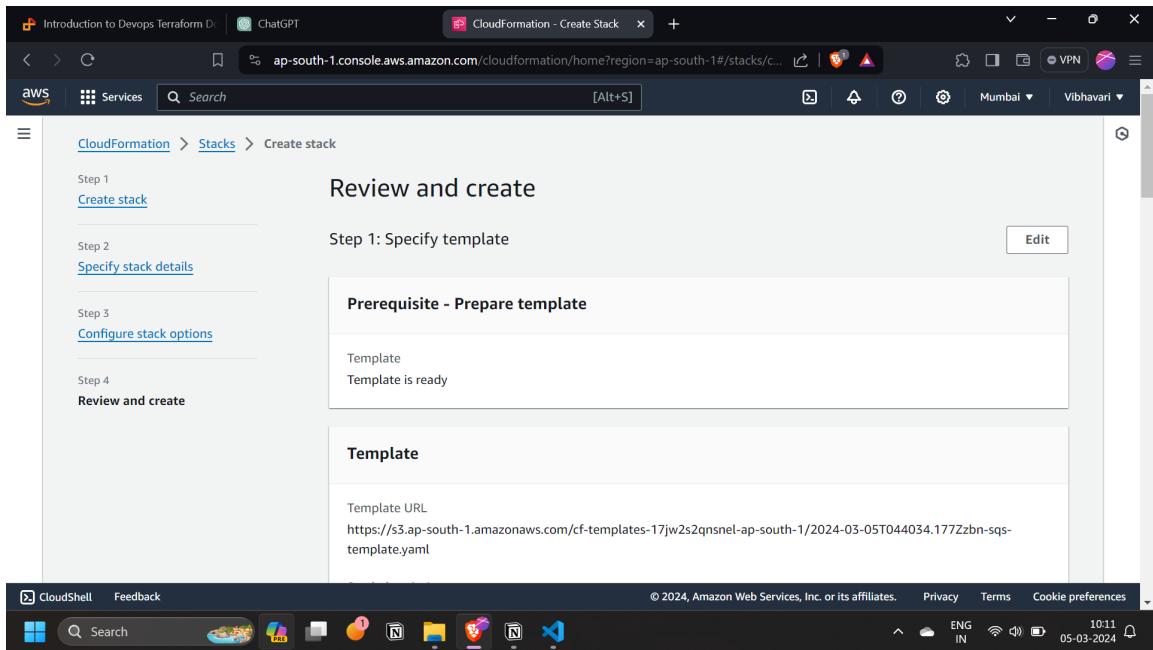
- Click "Next" to proceed to the stack creation wizard.



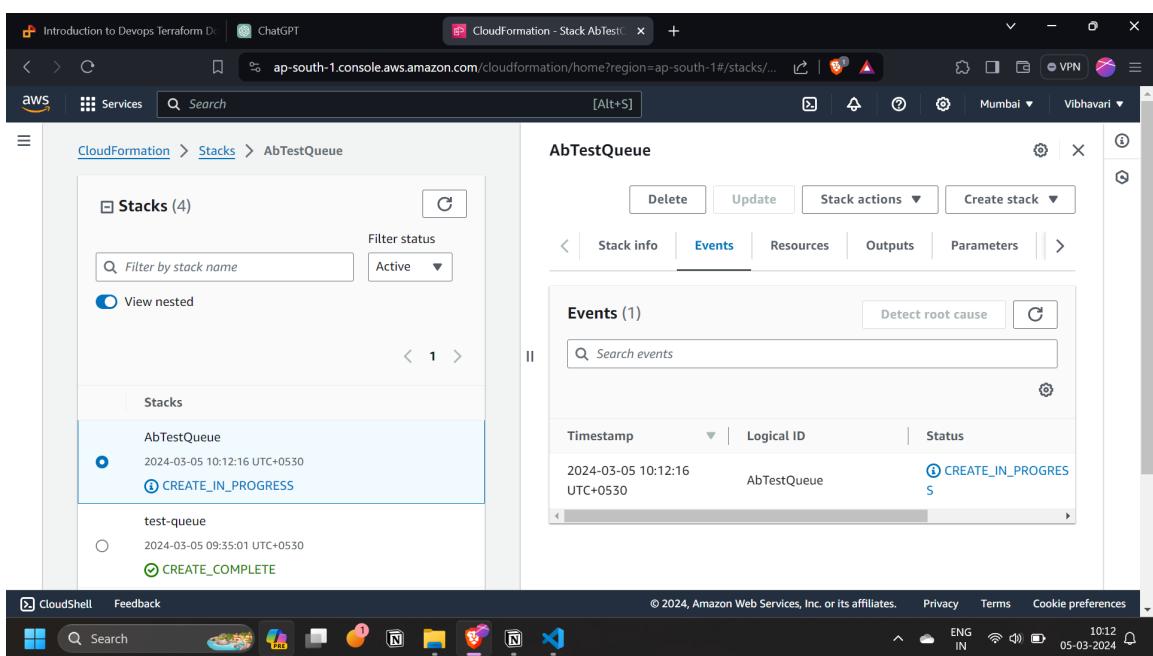
- Enter a stack name and any other parameters as needed.



- Click "Next" to proceed to the review page.



- Review the stack details and click "Create stack" to deploy the CloudFormation stack.



Test with Dummy Data:

- After the stack creation is complete, navigate to the SQS service in the AWS Management Console.

The screenshot shows the AWS Simple Queue Service (SQS) console. In the top navigation bar, there are tabs for 'Introduction to Devops Terraform D...', 'ChatGPT', and 'Queues | Simple Queue Service'. The main content area is titled 'Amazon SQS > Queues' and displays a table of three queues:

Name	Type	Created	Messages available	Messages in flight	Encryption
AbTestQueue	Standard	2024-03-05T10:12+05:30	0	0	Amazon SQS key (SSE-SQS)
MyTestQueue	Standard	2024-03-05T09:35+05:30	1	0	Amazon SQS key (SSE-SQS)
yash-queue-1	Standard	2024-03-05T10:04+05:30	0	0	Amazon SQS key (SSE-SQS)

At the bottom of the page, there is a footer with links for CloudShell, Feedback, and various AWS services like Lambda, S3, and CloudWatch. The footer also includes copyright information, privacy terms, and cookie preferences.

- Find the SQS queue created by the CloudFormation stack

The screenshot shows the 'Queue details' page for the 'AbTestQueue'. The top navigation bar is identical to the previous screenshot. The main content area is titled 'AbTestQueue' and shows the following details:

Name	Type	ARN
AbTestQueue	Standard	arn:aws:sqs:ap-south-1:905418082383:AbTestQueue

Below the table, there are sections for 'Encryption' (Amazon SQS key (SSE-SQS)) and 'URL' (<https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue>). A 'Dead-letter queue' section is also present with a link to 'None'. At the bottom of the page, there are tabs for 'SNS subscriptions', 'Lambda triggers', 'EventBridge Pipes', 'Dead-letter queue', 'Monitoring', 'Tagging', 'Access policy', and 'Encryption'. The footer is identical to the previous screenshot.

- Click on "Send a message".

The screenshot shows the AWS SQS 'Send and receive messages' interface. At the top, there are tabs for 'Send and receive messages' and 'Info'. Below the tabs, there is a 'Message body' input field containing 'Enter message'. A 'Delivery delay' field is set to '0 Seconds'. A 'Send message' button is visible at the bottom right of the form area.

- Enter some dummy data and click "Send message".

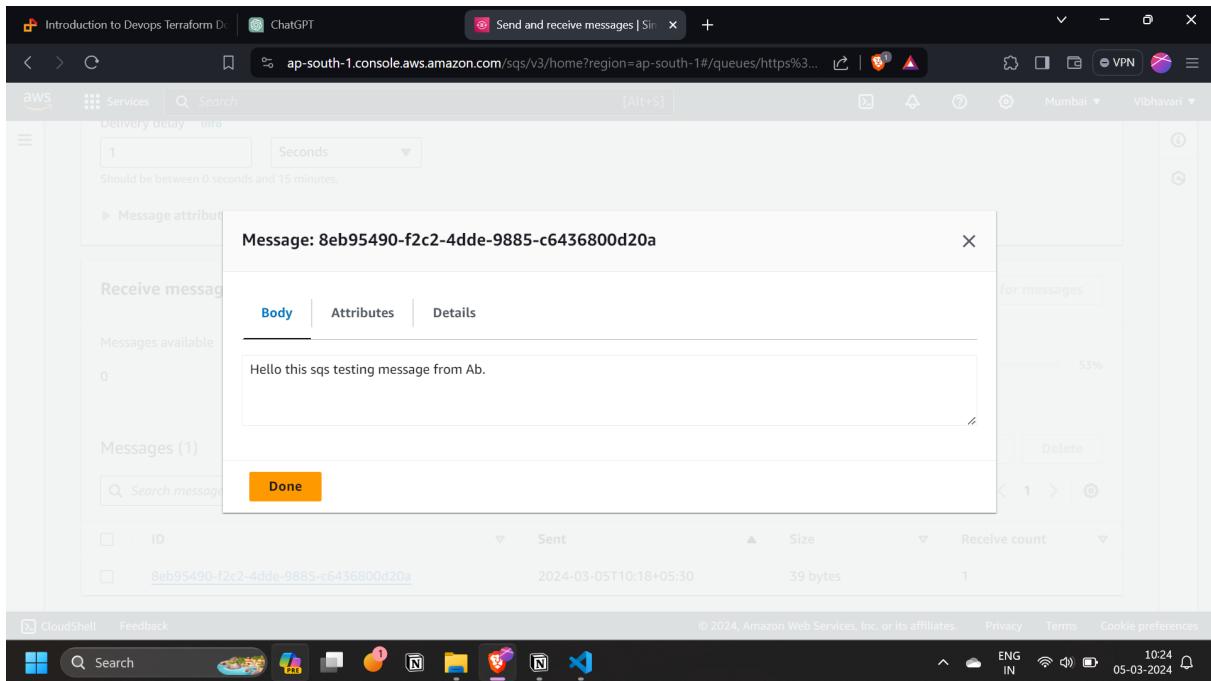
The screenshot shows the AWS SQS 'Send and receive messages' interface after a message has been sent. The 'Message body' input field now contains 'Hello this sqs testing message from Ab.'. The 'Delivery delay' field is set to '1 Seconds'. The 'Send message' button is present at the bottom right.

Verify message

- Verify that the message is successfully sent to the queue.

The screenshot shows the AWS SQS 'Send and receive messages' interface. At the top, there are tabs for 'Send and receive messages' and 'View details'. A success message box displays: 'Your message has been sent and can be received in 1 seconds.' Below this, the 'Message body' field contains the text 'Hello this sqs testing message from Ab.'. Under 'Delivery delay', the value '1' is selected in the dropdown. The status bar at the bottom indicates 'ENG IN' and the date '05-03-2024'.

The screenshot shows the AWS SQS 'Receive messages' interface. It displays a single message with ID '8eb95490-f2c2-4dde-9885-c6436800d20a', Sent on '2024-03-05T10:18+05:30', and Size '39 bytes'. The polling progress is at 13% with '1 receives/second'. The status bar at the bottom indicates 'ENG IN' and the date '05-03-2024'.



Testing Dummy Data using AWS CLI

Sending dummy data to the queue using the AWS CLI:

Command

```
aws sqs send-message --queue-url <https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue> --message-body '{"Name": "Abc", "age": 20}'
```

Screenshot

```

[cloudshell-user@ip-10-132-89-251 ~]$ aws sqs send-message --queue-url https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue --message-body '{"Name": "Abc", "age": 20}'
{
    "MD5OfMessageBody": "7291fc20c7c7859cc8c783ededa70fd4",
    "MessageId": "14ebc449-9317-4422-b308-a994030c6e86"
}
[cloudshell-user@ip-10-132-89-251 ~]$ aws sqs receive-message --queue-url https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue
{
    "Messages": [
        {
            "MessageId": "007e21f8-b8a2-4255-baf3-3e012d439fb8",
            "ReceiptHandle": "AQEB9FHGbfzpb8YAKIPjNWSZC15VwgFK+MvHQ8t5JRpLxbjCB8716ocG1KBYfIowouD6vSqnKmHqoAicpMyTubBjM5/FcyjEgU13xds0AEPfkGwAN6Tjh9C+RpK2Is9FaD5LCH4eh7/8+nLJy3wP51JiPFNSvx12APkD+1s3DQnRkSmynHfEZfTm81ZUfahp2P4ubTCVuL3JOWJEGfHR1ldcsxmHDEESf1SFH8ek59ZAYtmvTKNqHr9LJrhLaJCKB2UYPFz8CwHGx9W5KDj0JeSys+gWlNBnETtiLnri1ETByG8XLk9SyrcPJrMkbzPAanseonSc3cd42/XV4GNEXkRkyEHqo1guvgkKc96dwGQ1yVlp/u/itYhPVTfFyzqYc5enQ==",
            "MD5OfBody": "7291fc20c7c7859cc8c783ededa70fd4",
            "Body": "{\"Name\": \"Abc\", \"age\": 20}"
        }
    ]
}
[cloudshell-user@ip-10-132-89-251 ~]$

```

View the message in AWS CLI:

Command:

```
aws sqs receive-message --queue-url <https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue>
```

Screenshot

```

[cloudshell-user@ip-10-132-89-251 ~]$ aws sqs send-message --queue-url https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue --message-body '{"Name": "Abc", "age": 20}'
{
    "MD5OfMessageBody": "7291fc20c7c7859cc8c783ededa70fd4",
    "MessageId": "14ebc449-9317-4422-b308-a994030c6e86"
}
[cloudshell-user@ip-10-132-89-251 ~]$ aws sqs receive-message --queue-url https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue
{
    "Messages": [
        {
            "MessageId": "007e21f8-b8a2-4255-baf3-3e012d439fb8",
            "ReceiptHandle": "AQEB9FHGbfzpb8YAKIPjNWSZC15VwgFK+MvHQ8t5JRpLxbjCB8716ocG1KBYfIowouD6vSqnKmHqoAicpMyTubBjM5/FcyjEgU13xds0AEPfkGwAN6Tjh9C+RpK2Is9FaD5LCH4eh7/8+nLJy3wP51JiPFNSvx12APkD+1s3DQnRkSmynHfEZfTm81ZUfahp2P4ubTCVuL3JOWJEGfHR1ldcsxmHDEESf1SFH8ek59ZAYtmvTKNqHr9LJrhLaJCKB2UYPFz8CwHGx9W5KDj0JeSys+gWlNBnETtiLnri1ETByG8XLk9SyrcPJrMkbzPAanseonSc3cd42/XV4GNEXkRkyEHqo1guvgkKc96dwGQ1yVlp/u/itYhPVTfFyzqYc5enQ==",
            "MD5OfBody": "7291fc20c7c7859cc8c783ededa70fd4",
            "Body": "{\"Name\": \"Abc\", \"age\": 20}"
        }
    ]
}
[cloudshell-user@ip-10-132-89-251 ~]$

```

Task 3: Install Docker on the EC2 Instance

1. Connect to the EC2 Instance:

- Log in to the AWS Management Console.
- Navigate to the EC2 service.
- Select the EC2 instance that you previously created.
- Click on "Connect" to view connection instructions.
- Follow the instructions to connect to the instance using SSH.

2. Update the Package Repository:

- Once connected to the EC2 instance, update the package repository to ensure you install the latest version of Docker.

```
sudo apt update
```

3. Install Docker:

- Install Docker CE (Community Edition) using the package manager.

```
sudo apt install docker.io
```

4. Start and Enable Docker Service:

- Start the Docker service and enable it to start on boot.

```
sudo systemctl start docker
sudo systemctl enable docker
```

5. Verify Docker Installation:

- Check the Docker version to ensure that it's installed correctly.

```
docker --version
```

6. Test Docker:

- Run a simple Docker command to verify that Docker is installed and working properly.

```
sudo docker run hello-world
```

This command should download and run the "hello-world" Docker image. If everything is set up correctly, you should see a message confirming that Docker is working.

Conclusion

By completing these steps, you have successfully installed Docker on the EC2 instance. This will allow you to create and manage Docker containers on your instance, which can be useful for various development and deployment tasks.

Task 4 Create IAM Role with Limited Access

Create IAM Role with Limited Access (ECR Permissions should be allowed)

Description: This subtask involves creating an IAM role that grants limited permissions, specifically allowing access to Amazon Elastic Container Registry (ECR) resources.

Here's a CloudFormation template to create an IAM role with limited access, specifically granting permissions for Amazon ECR (Elastic Container Registry):

```
AWS::TemplateFormatVersion: '2010-09-09'
```

```
Resources:
```

```
    ECRRole:
```

```

Type: AWS::IAM::Role
Properties:
  RoleName: ECRRole
  AssumeRolePolicyDocument:
    Version: '2012-10-17'
    Statement:
      - Effect: Allow
        Principal:
          Service: ec2.amazonaws.com
        Action: sts:AssumeRole
  Policies:
    - PolicyName: ECRPolicy
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - ecr:GetAuthorizationToken
              - ecr:BatchCheckLayerAvailability
              - ecr:GetDownloadUrlForLayer
              - ecr:GetRepositoryPolicy
              - ecr:DescribeRepositories
              - ecr>ListImages
              - ecr:DescribeImages
              - ecr:BatchGetImage
      Resource: '*'

```

Explanation:

- This CloudFormation template defines an IAM role named `ECRRole`.
- The `AssumeRolePolicyDocument` specifies that only EC2 instances (`ec2.amazonaws.com`) are allowed to assume this role.
- The `ECRPolicy` grants permissions for various actions within Amazon ECR, such as `GetAuthorizationToken`, `BatchCheckLayerAvailability`, etc.
- The `Resource: '*'` allows access to all resources within ECR. You can further restrict this to specific repositories if needed.

We will deploy this template using the AWS Management Console, AWS CLI, or any other AWS SDK that supports CloudFormation. Make sure to replace the `RoleName` with a suitable name for your use case.

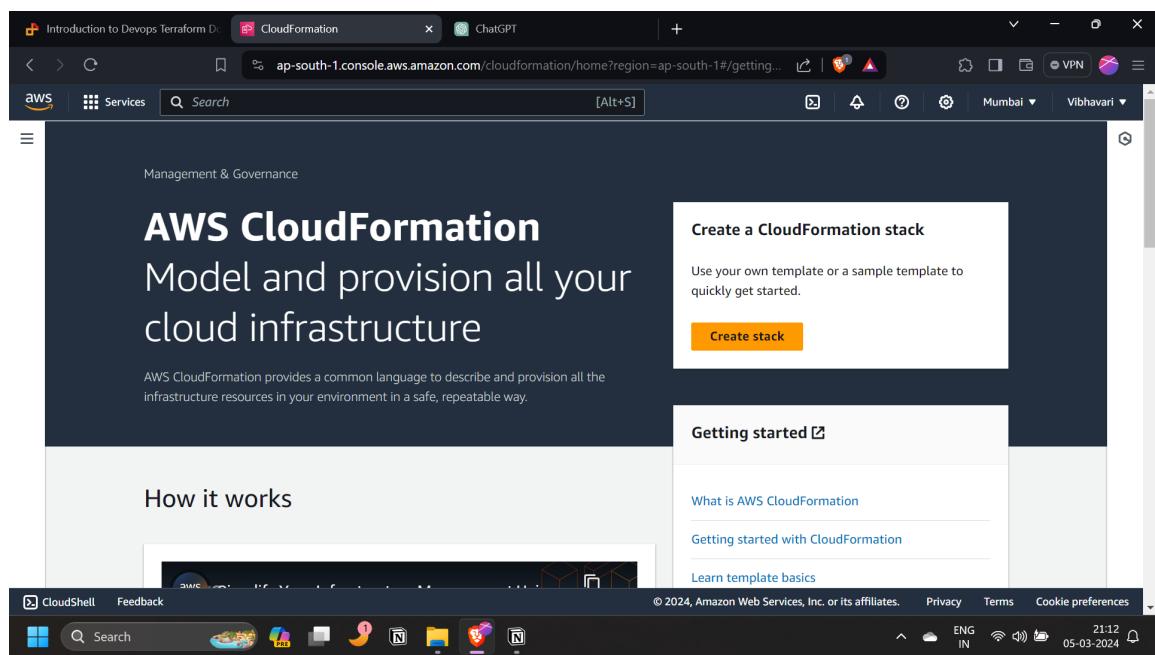
Implement the IAM role with limited access for Amazon ECR using the provided CloudFormation:

1. Save the CloudFormation Template:

Save the CloudFormation template provided in the previous response to a file with a `.yaml` or `.json` extension. You can name it something like `ecr-role-template.yaml`.

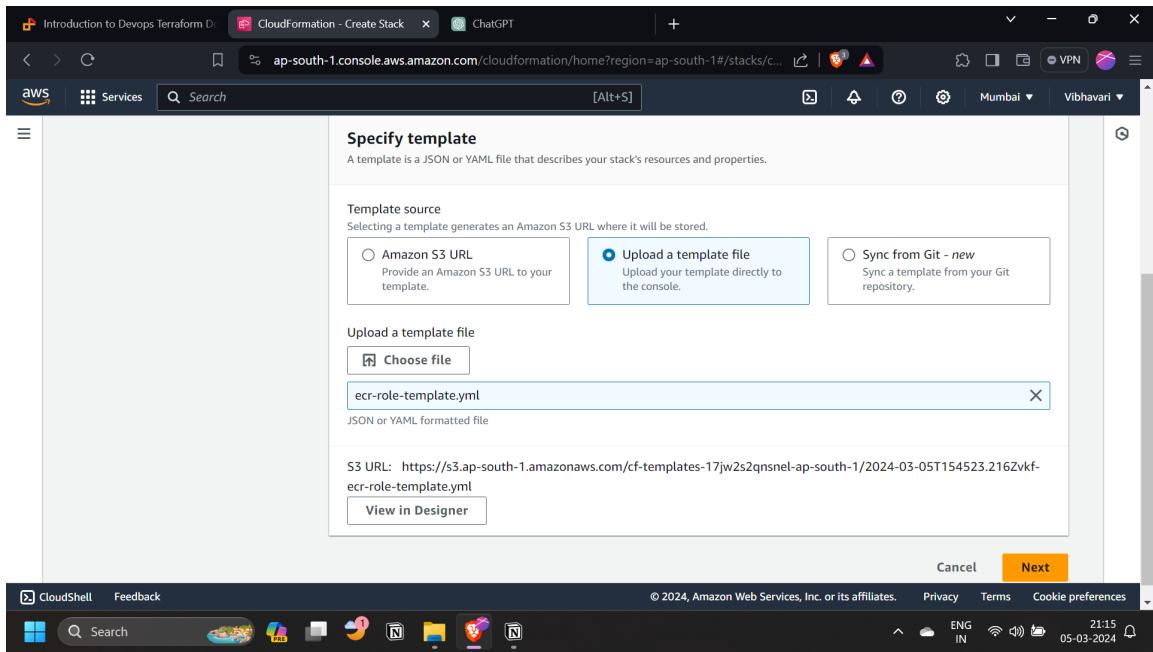
2. Navigate to CloudFormation:

Go to the AWS CloudFormation service by either searching for it in the AWS services search bar or navigating through the services dropdown.



3. Select Template:

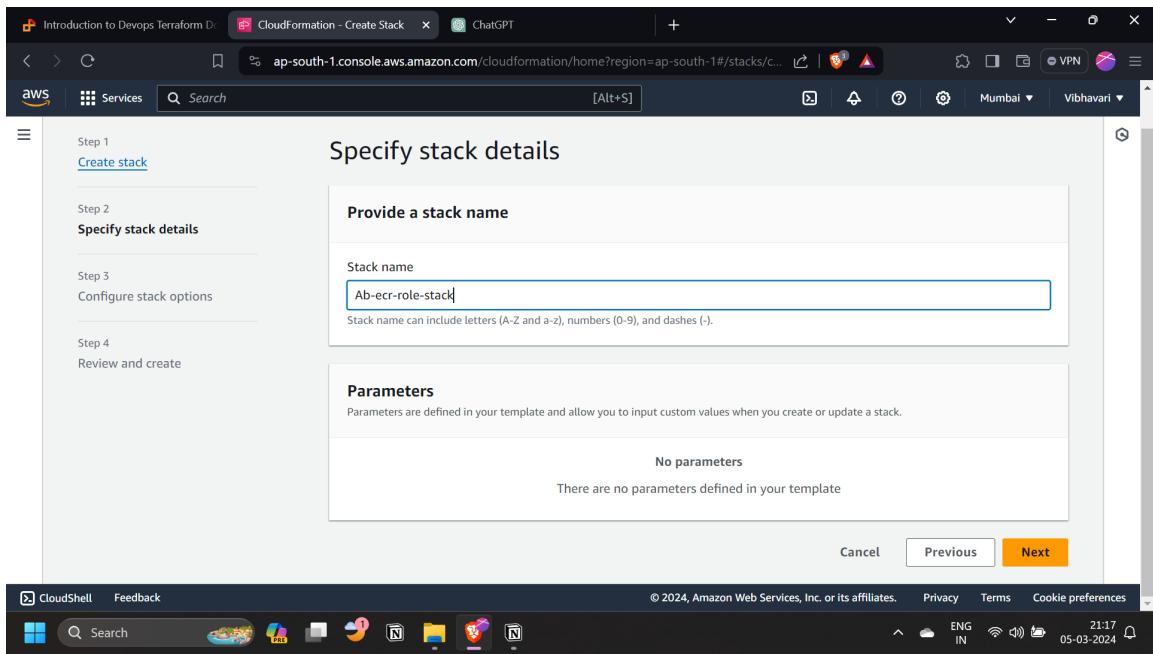
Choose "Template is ready" and then select "Upload a template file". Click on "Choose file" and select the CloudFormation template file (`ecr-role-template.yaml`) you saved earlier.



4. Specify Stack Details:

Provide a stack name, for example,

`ECRRoleStack`.

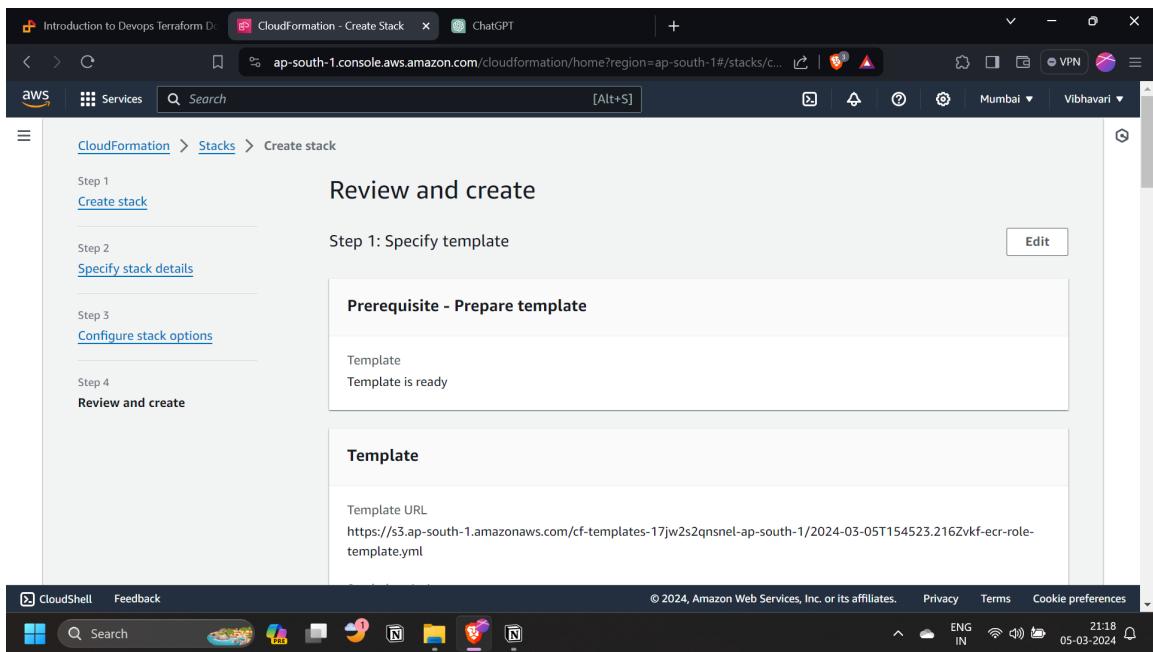


5. Configure Stack Options (Optional):

You can specify additional configurations such as tags, permissions, etc., if needed. Otherwise, you can leave these as default.

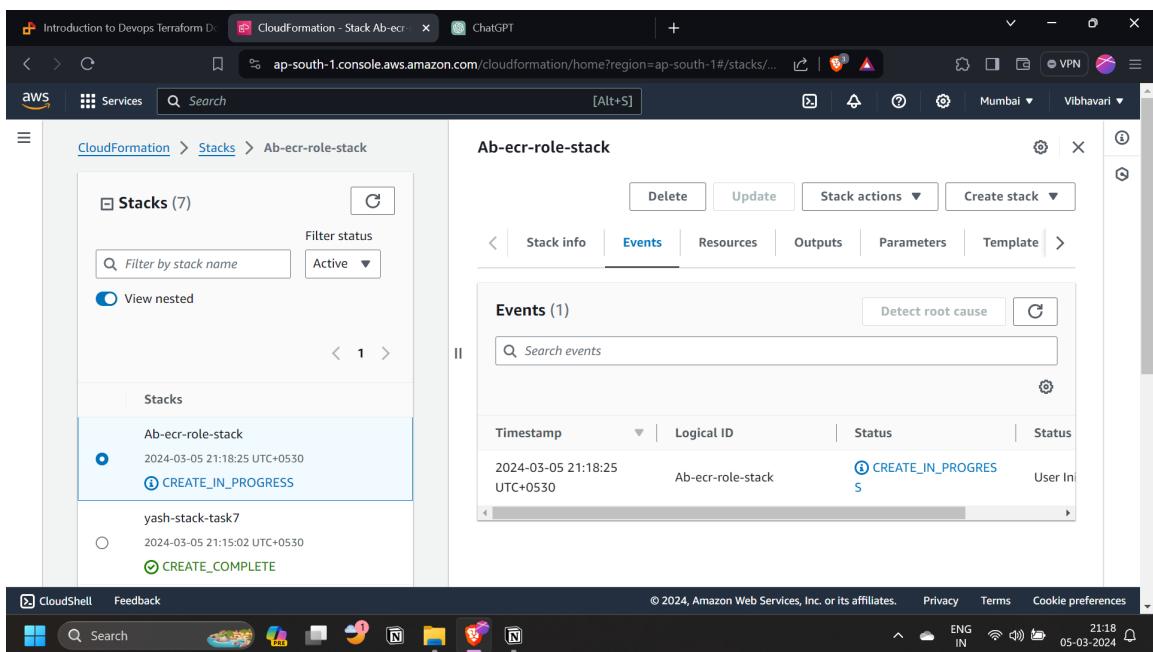
6. Review and Create:

Review the stack details and click on "Create stack" to initiate the creation process.



7. Wait for Stack Creation:

Wait for the CloudFormation stack creation process to complete. This usually takes a few minutes.



8. Verify Stack Creation:

Once the stack creation is complete, navigate to the "Stacks" section in the CloudFormation console. Verify that the stack status is "CREATE_COMPLETE".

The screenshot shows the AWS CloudFormation console with the 'Events' tab selected for the stack 'Ab-ecr-role-stack'. The event table displays one entry:

Timestamp	Logical ID	Status
2024-03-05 21:18:25 UTC+0530	Ab-ecr-role-stack	CREATE_IN_PROGRESS

9. Access the IAM Role:

You can now access the IAM role named

ECRRole that you've created. This role grants limited access to Amazon ECR resources as defined in the CloudFormation template.

The screenshot shows the AWS CloudFormation console with the 'Resources' tab selected for the stack 'Ab-ecr-role-stack'. The resource table displays one entry:

Logical ID	Physical ID	Type
ECRRole	ECRRole	AWS::IAM::Role

Conclusion:

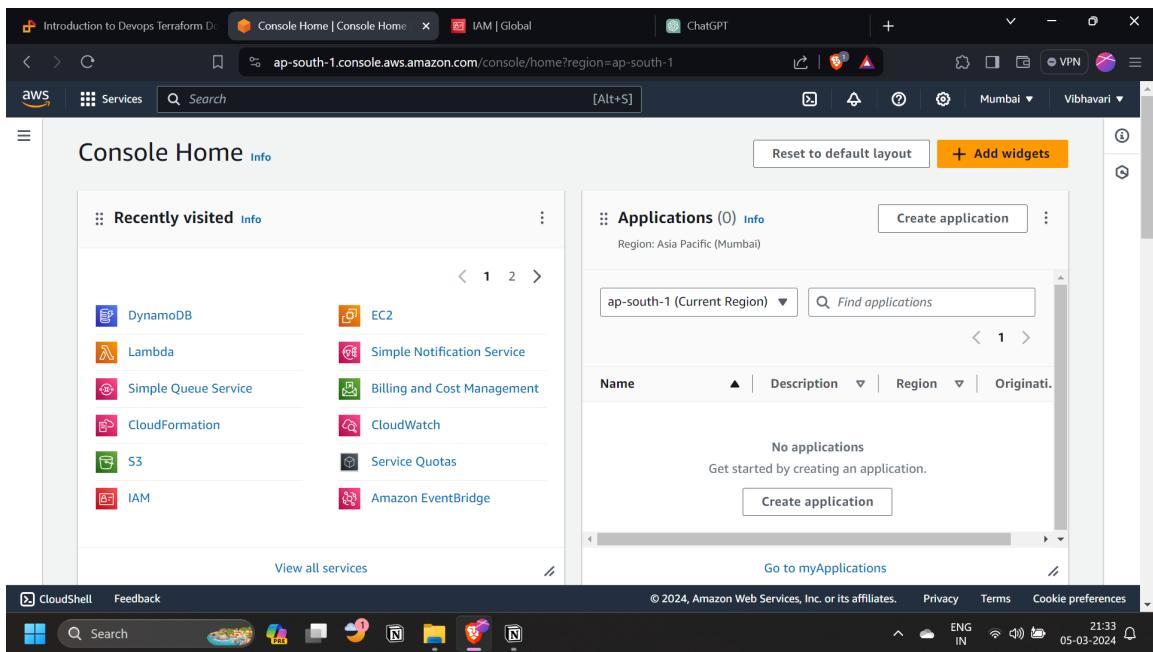
Implemented an IAM role with limited access for Amazon ECR using CloudFormation. This role, named ECRRole, grants specific permissions to interact with Amazon ECR resources, ensuring security and compliance in your AWS environment. We can now leverage this role for various use cases such as managing container images, deploying applications, or integrating with other AWS services, all while adhering to a least privilege access model.

Task 5: Attach IAM role to EC2 instance

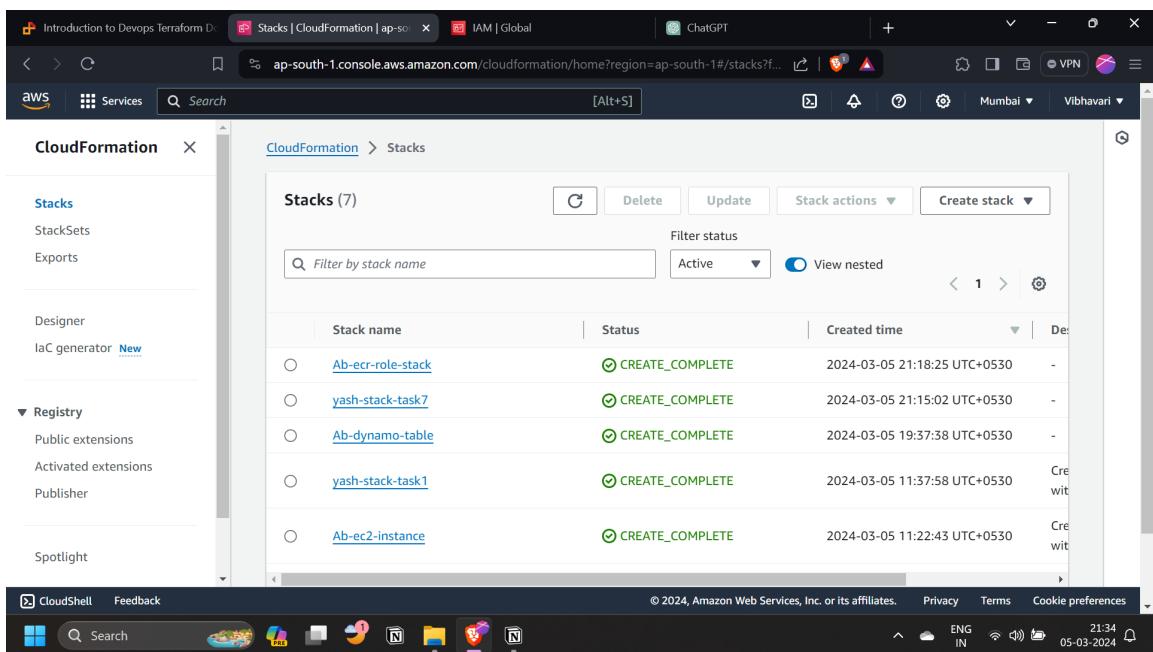
Attach IAM role to above created instance in Task 1.

Implementation of attaching an IAM role to an existing EC2 instance using the AWS Management Console:

- 1. Sign in to the AWS Management Console:** Go to the AWS Management Console and sign in using your credentials.



2. Navigate to the EC2 Dashboard: Once logged in, navigate to the EC2 dashboard by searching for "EC2" in the AWS services search bar and clicking on it.



The screenshot shows the AWS CloudFormation console with the following details:

- Stacks (7)**: The list of stacks includes:
 - yash-stack-task7
 - Ab-dynamo-table
 - yash-stack-task1
 - Ab-ec2-instance** (selected)
 - AbTestQueue
 - test-queue
- Status**: All stacks are in the **CREATE_COMPLETE** state.
- Created time**: Various dates between March 5, 2024, and March 6, 2024.

The screenshot shows the AWS CloudFormation console with the following details:

- Stacks (7)**: The list of stacks includes:
 - 2024-03-05 11:37:58 UTC+0530 (CREATE_COMPLETE)
 - Ab-ec2-instance** (selected)
 - 2024-03-05 10:12:16 UTC+0530 (CREATE_COMPLETE)
- Resources (2)**: The resources listed are:

Logical ID	Physical ID	Type
EC2Instance	i-03acd069d07393ba6	AWS::EC2::Instance
EC2SecurityGroup	Ab-ec2-instance_EC2SecurityGroup-AVKI74DZ7FH	AWS::EC2::SecurityGroup

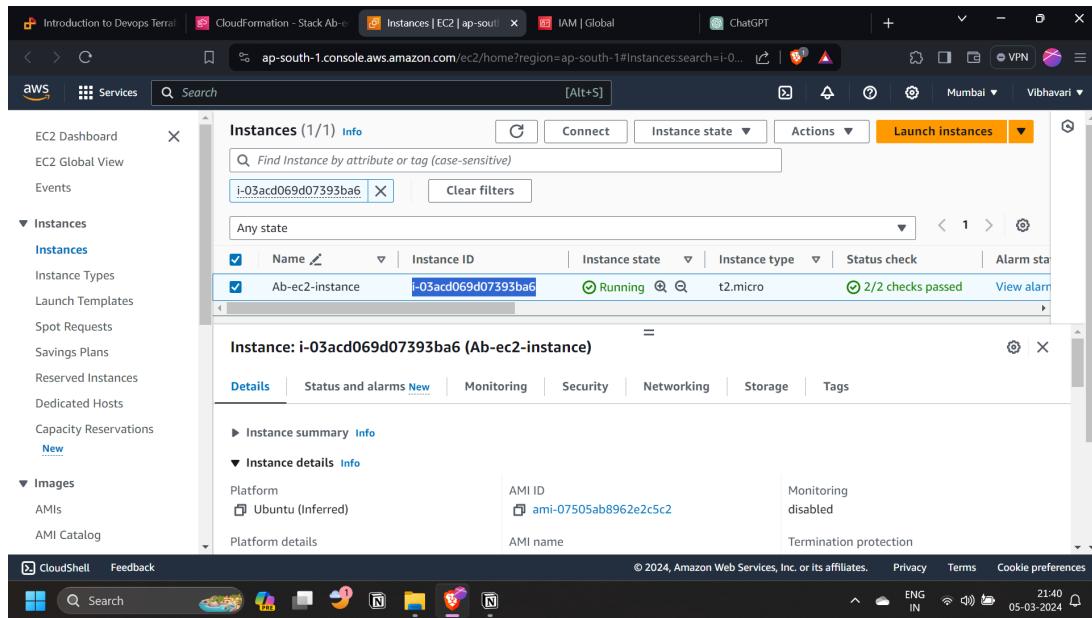
3. **Select the EC2 Instance:** From the EC2 dashboard, locate and select the Ubuntu EC2 instance that you created in Task 1. Click on the instance ID to select it.

The screenshot shows the AWS EC2 Instances page. The main header includes tabs for 'Instances | EC2 | ap-south-1' and 'CloudFormation - Stack Ab'. The search bar contains 'ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#Instances;search=i-0...'. The top navigation bar has links for 'EC2 Dashboard', 'Services', 'Search', and 'Actions'. A 'Launch instances' button is prominently displayed. On the left, a sidebar lists 'Instances' (selected), 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Capacity Reservations', and 'Images'. The main content area shows 'Instances (1/1) Info' with a table. The table has columns for 'Name' (sorted), 'Instance ID' (i-03acd069d07393ba6), 'Instance state' (Running), 'Instance type' (t2.micro), 'Status check' (2/2 checks passed), and 'Alarm status'. Below the table, the 'Instance: i-03acd069d07393ba6 (Ab-ec2-instance)' details are shown, including 'Details', 'Status and alarms New', 'Monitoring', 'Security', 'Networking', 'Storage', and 'Tags' tabs. The instance summary shows it's running on an Ubuntu (Inferred) platform with AMI ID ami-07505ab8962e2c5c2. The monitoring status is disabled. The bottom of the page includes a footer with links for 'CloudShell', 'Feedback', and various AWS services, along with system status icons.

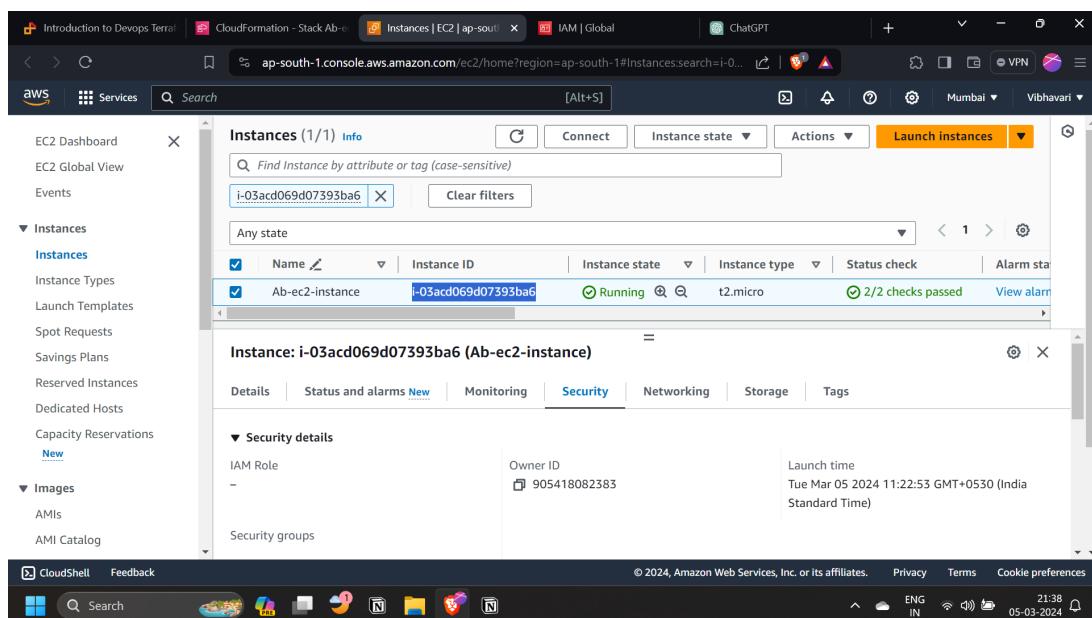
This screenshot is identical to the one above, showing the AWS EC2 Instances page with a single running t2.micro instance named 'Ab-ec2-instance'. The interface, data, and footer are exactly the same.

4. Attach IAM Role:

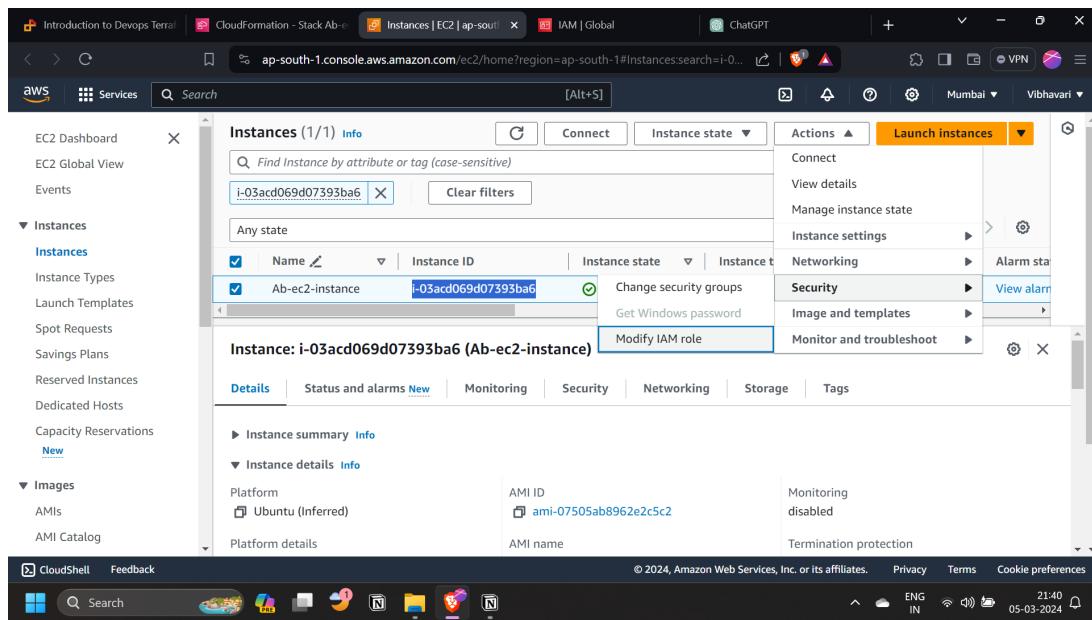
- With the EC2 instance selected, scroll down to the "Description" tab at the bottom of the page.



- In the "Details" section, find the "IAM role" field, which initially may show "None".



- Click on the "Actions" dropdown button and select "Security" > "Modify IAM role".



- In the "Modify IAM role" dialog, you'll see a dropdown list of available IAM roles. Select the IAM role named `ECRRole` that you created in Task 4.
- Click on the "Apply" button to attach the IAM role to the EC2 instance.

5. Confirmation:

- After attaching the IAM role, you'll receive a confirmation message indicating that the IAM role has been successfully attached to the EC2 instance.

6. Verify Attachment:

- To verify that the IAM role has been attached, you can select the EC2 instance again and check the "IAM role" field in the "Description" tab. It should now display the name of the attached IAM role (`ECRRole`).

That's it! You have successfully attached the IAM role named `ECRRole` to the Ubuntu EC2 instance created earlier. Now the instance has the necessary permissions to interact with Amazon ECR, as defined by the IAM role.

Task 6: Create DynamoDB Table

Creating DynamoDB table with AWS Management Console:

Using AWS Management Console:

- Go to the AWS Management Console and navigate to the DynamoDB service.
- Click on "Create table".
- Enter the table name (e.g., YourTableName).
- Under "Primary key", enter "ProductID" as the partition key and select "String" as its type.
- Add additional attributes by clicking on "Add sort key" or "Add index" depending on your requirements.
- Click on "Create".

Using AWS CloudFormation:

Below is a CloudFormation template to create a DynamoDB table:

```
Resources:  
  ProductTable:  
    Type: 'AWS::DynamoDB::Table'  
    Properties:  
      TableName: Product  
      AttributeDefinitions:  
        - AttributeName: productId  
          AttributeType: S  
      KeySchema:  
        - AttributeName: productId  
          KeyType: HASH  
      ProvisionedThroughput:  
        ReadCapacityUnits: 5  
        WriteCapacityUnits: 5
```

To deploy this template:

- Save the above YAML content in a file, say `dynamodb-table.yaml`.
- Go to the AWS CloudFormation service in the AWS Management Console.
- Click on "Create stack" and upload the YAML file.

- Follow the prompts to create the stack, providing necessary inputs like stack name.
- Once the stack creation is complete, your DynamoDB table will be created according to the specified configuration.

Implementation of Creating Dynamodb table using Cloudformation

Creating a DynamoDB table using AWS CloudFormation:

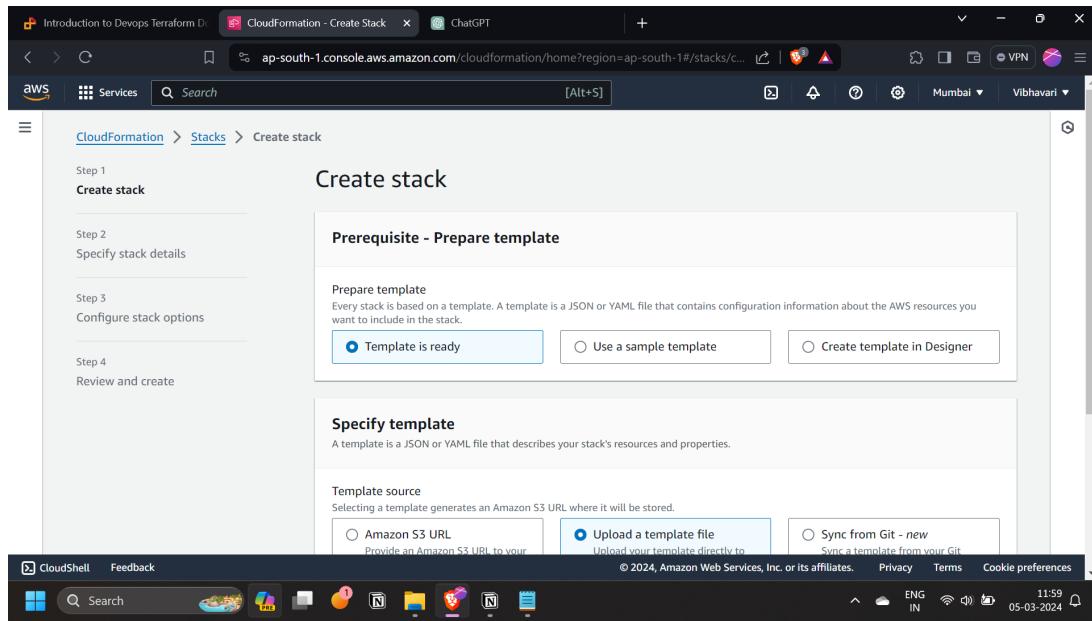
1. Authoring the CloudFormation Template:

Template

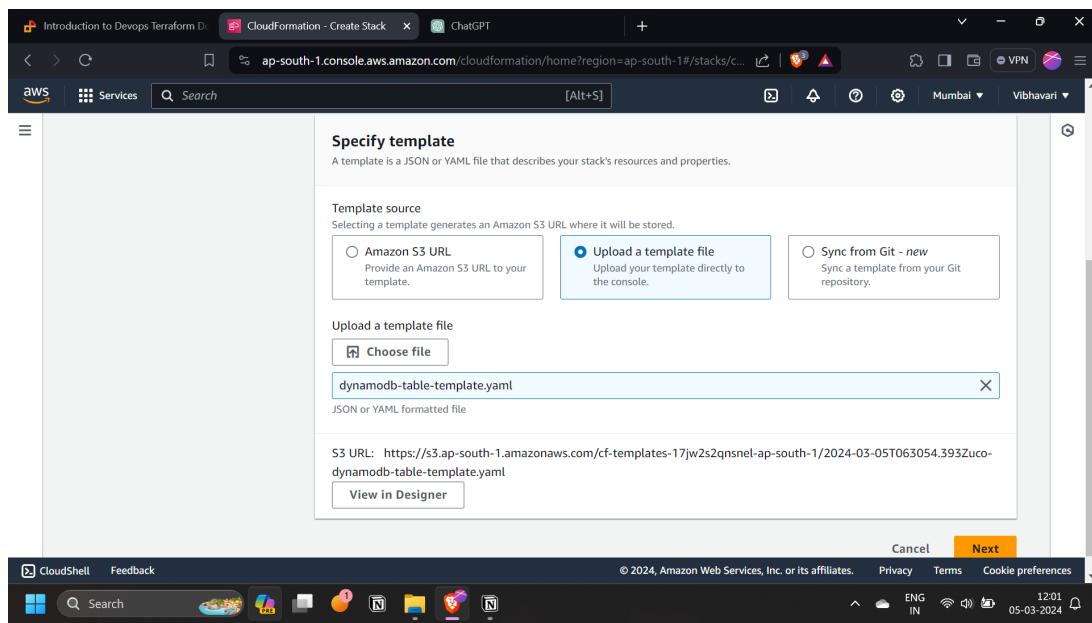
```
Resources:
  ProductTable:
    Type: 'AWS::DynamoDB::Table'
    Properties:
      TableName: Product
      AttributeDefinitions:
        - AttributeName: productId
          AttributeType: S
      KeySchema:
        - AttributeName: productId
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
```

2. Deploying the CloudFormation Stack:

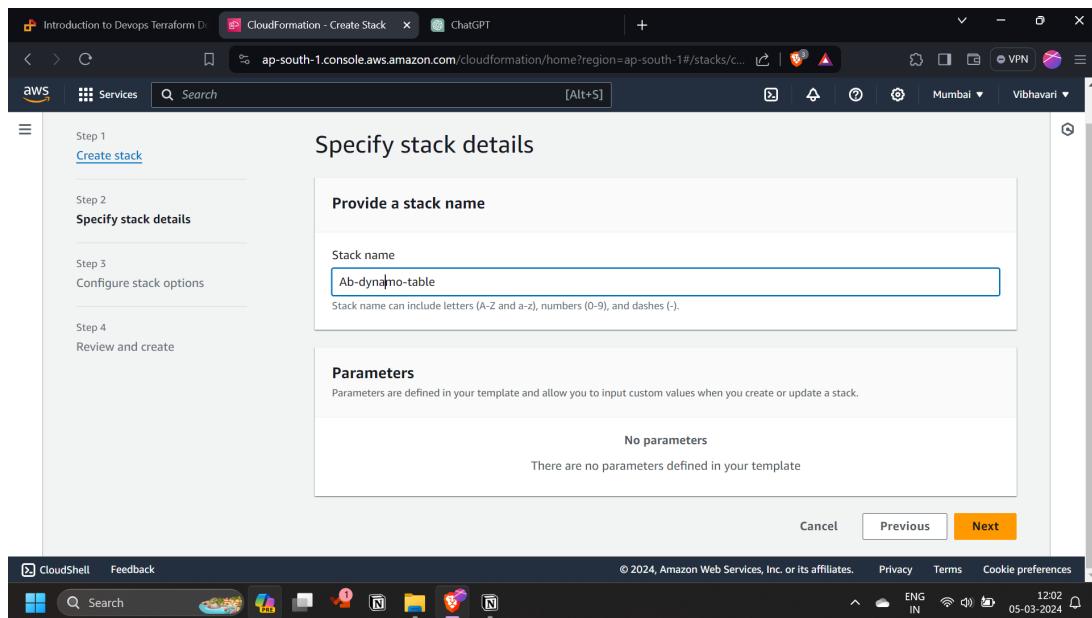
- Click on "Create stack".



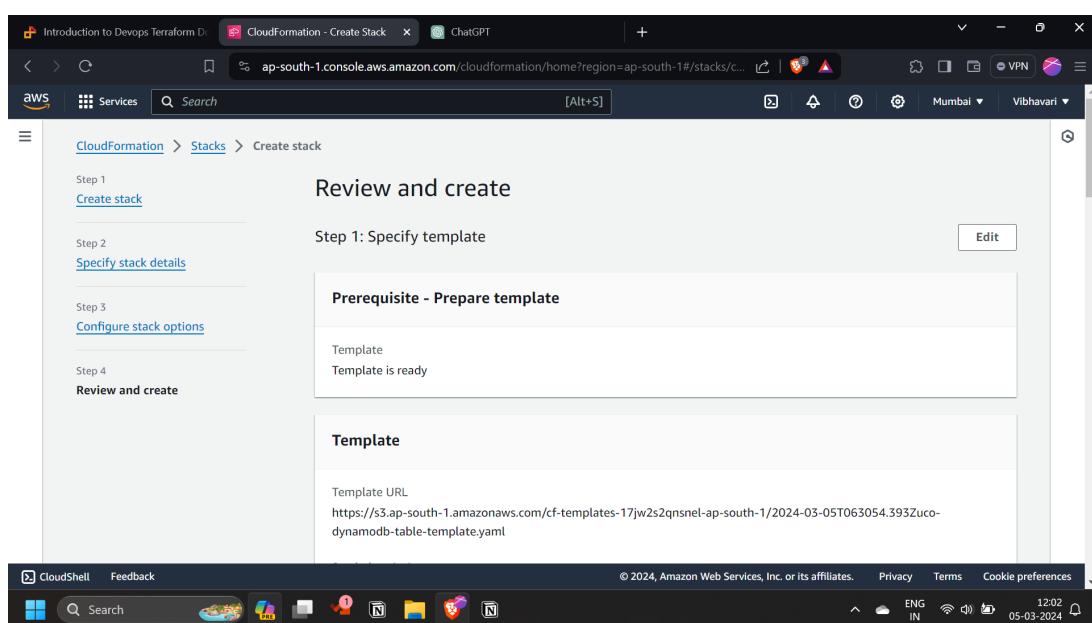
- Choose "Template is ready" and "Upload a template file", then select the template file you created.



- Enter a stack name.



- Review the configuration and click "Create stack".



3. Monitoring Stack Creation:

CloudFormation will now create the stack. You can monitor its progress in the CloudFormation dashboard. Once the stack creation is complete, your DynamoDB table will be provisioned according to the specifications in your CloudFormation template.

The screenshot shows the AWS CloudFormation console with the URL ap-south-1.console.aws.amazon.com/cloudformation/home?region=ap-south-1#/stacks/. The left sidebar shows a list of stacks, with 'Ab-dynamo-table' selected. The main panel displays the details for the 'Ab-dynamo-table' stack, specifically the 'Events' tab which lists one event: 'CREATE_IN_PROGRESS' at 2024-03-05 12:03:08 UTC+0530.

The screenshot shows the AWS CloudFormation console with the same URL as the previous screenshot. The left sidebar shows a list of stacks, with 'Ab-dynamo-table' selected. The main panel displays the details for the 'Ab-dynamo-table' stack, specifically the 'Events' tab which lists one event: 'CREATE_COMPLETE' at 2024-03-05 19:37:38 UTC+0530.

Verification:

After the stack creation is successful, navigate to the DynamoDB service in the AWS Management Console and verify that your table `TableName` has been created with the specified attributes and settings.

Task 7: Creating Lambda Function

Implementation of Creating Lambda Function using Cloud formation template:

Sure, here are the important steps for implementing the CloudFormation template to create a Lambda function with DynamoDB and SQS access:

- 1. Prepare Your Template:** Modify the provided CloudFormation template by replacing `'YourDynamoDBTableName'` with your actual DynamoDB table name and `'YourSQSQueueURL'` with your actual SQS queue URL. Save the modified template with a `.yaml` extension.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template to create a Lambda function with DynamoDB and SQS access'

Resources:
  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Principal:
              Service: 'lambda.amazonaws.com'
              Action: 'sts:AssumeRole'
      Policies:
        - PolicyName: 'LambdaDynamoDBSQSPolicy'
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: 'Allow'
                Action:
                  - 'dynamodb:PutItem'
                Resource: '*'
              - Effect: 'Allow'
                Action:
                  - 'sns:SendMessage'
                Resource: 'arn:aws:sns:ap-south-1:905418'
```

```

082383:ab-queue'

LambdaFunction:
  Type: 'AWS::Lambda::Function'
Properties:
  Handler: 'index.lambda_handler'
  Role: !GetAtt LambdaExecutionRole.Arn
  Code:
    ZipFile: |
      import boto3
      import json
      import time

      dynamodb = boto3.client('dynamodb')
      sqs = boto3.client('sns')

      table_name = 'YourDynamoDBTableName'
      queue_url = 'YourSQSQueueURL'
      max_retries = 3

    def lambda_handler(event, context):
        try:
            data = json.loads(event['body'])

            response = dynamodb.put_item(
                TableName=table_name,
                Item={
                    'id': {'S': data['id']},
                    'payload': {'S': json.dumps(da
ta)}
                }
            )

            sqs.send_message(
                QueueUrl=queue_url,
                MessageBody=json.dumps(data)
            )

```

```

        return {
            'statusCode': 200,
            'body': json.dumps('Data stored in
DynamoDB and sent to SQS successfully')
        }
    except Exception as e:
        retries = 0
        while retries < max_retries:
            retries += 1
            time.sleep(retries * 2)
        try:
            response = lambda_handler(event,
t, context)
            return response
        except Exception as e:
            if retries == max_retries:
                return {
                    'statusCode': 500,
                    'body': json.dumps('Fa
iled to store data in DynamoDB and send to SQS after max
imum retries')
                }
            else:
                continue

```

Runtime: python3.8

Timeout: 30

LambdaPermissionDynamoDB:

Type: 'AWS::Lambda::Permission'

Properties:

Action: 'lambda:InvokeFunction'

FunctionName: !GetAtt LambdaFunction.Arn

Principal: 'dynamodb.amazonaws.com'

LambdaPermissionSQS:

Type: 'AWS::Lambda::Permission'

Properties:

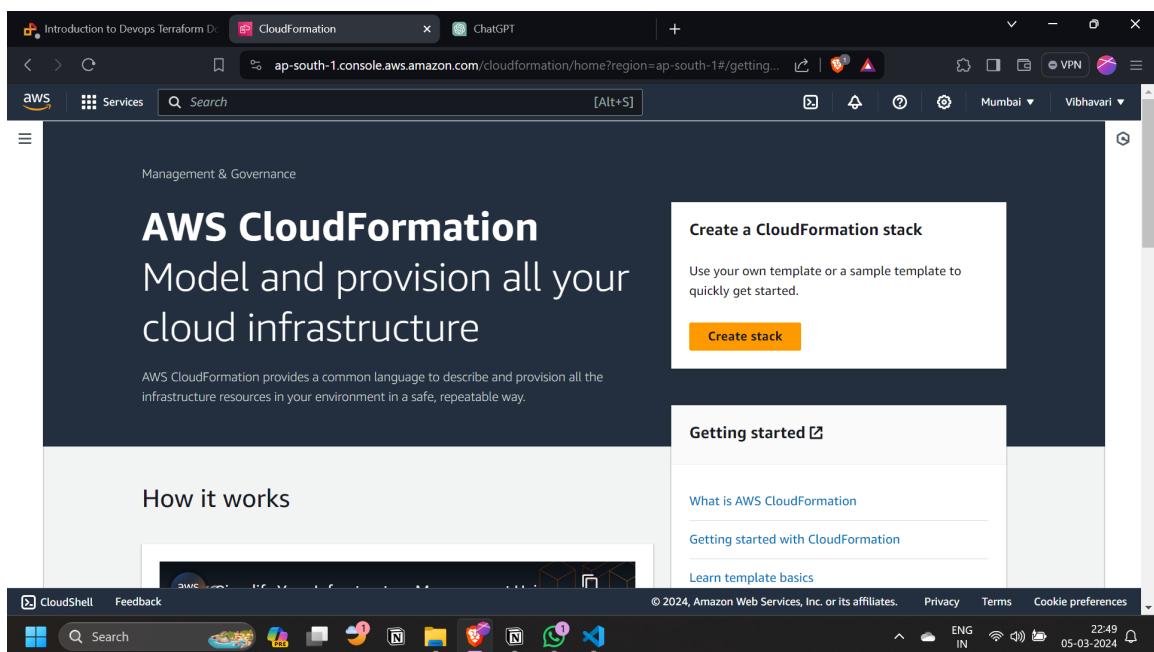
```
Action: 'lambda:InvokeFunction'  
FunctionName: !GetAtt LambdaFunction.Arn  
Principal: 'sns.amazonaws.com'
```

Outputs:

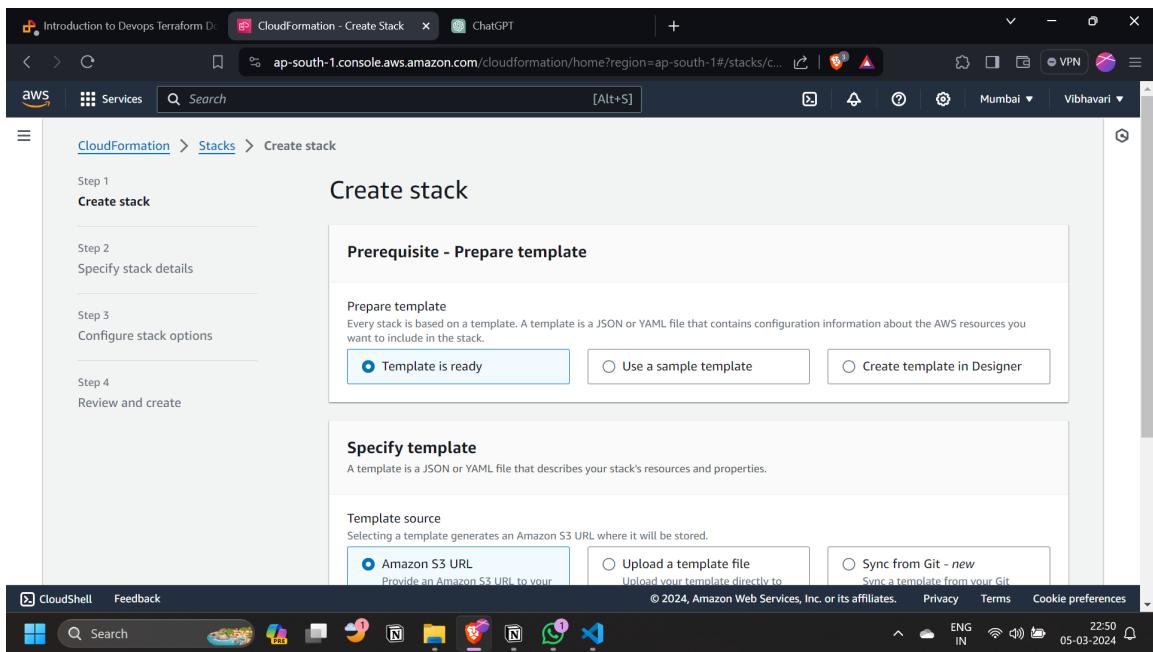
LambdaFunctionARN:

```
Description: 'ARN of the created Lambda function'  
Value: !GetAtt LambdaFunction.Arn
```

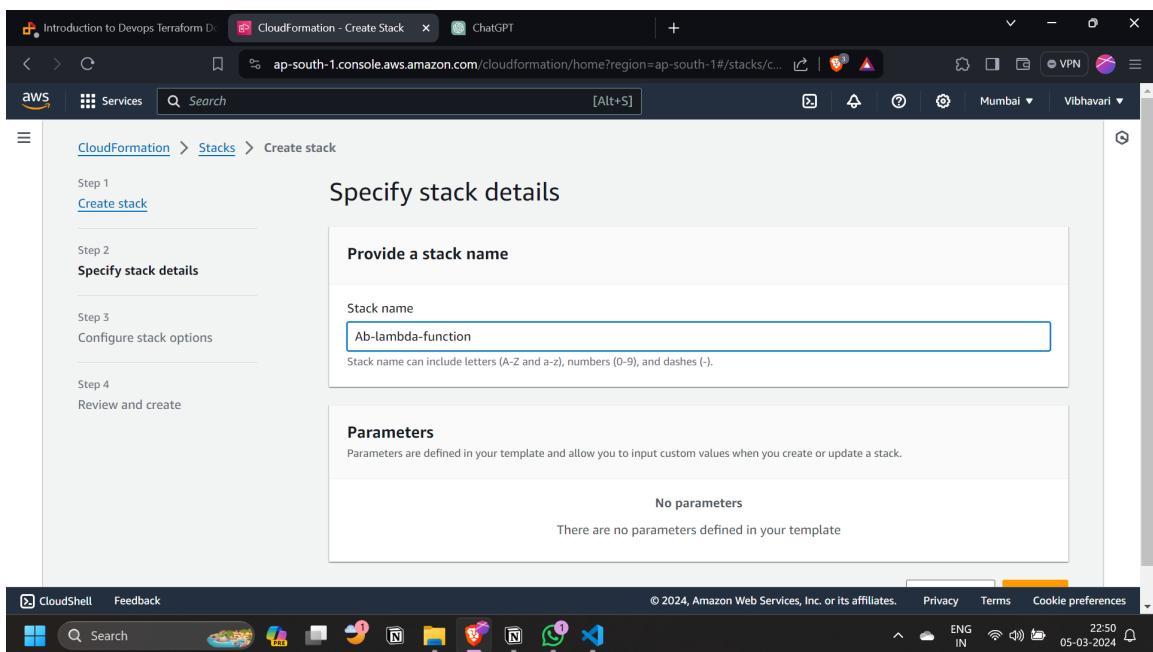
2. **Navigate to CloudFormation:** Go to the AWS Management Console and navigate to the CloudFormation service.



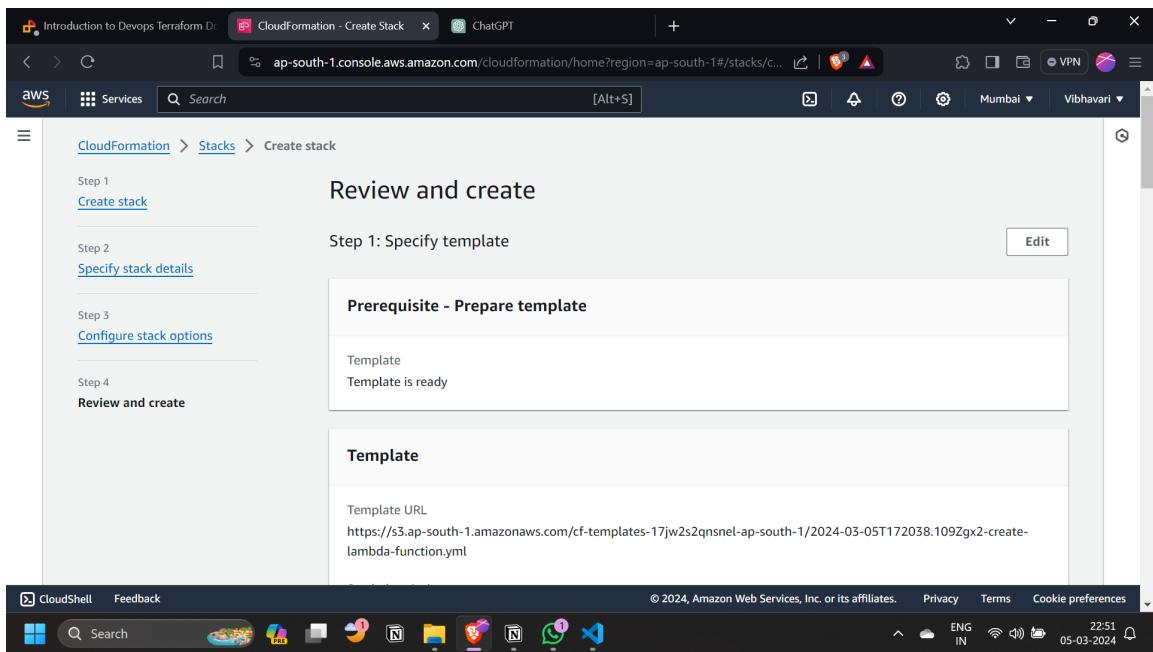
3. **Create a New Stack:** Click on "Create stack" and upload the modified CloudFormation template file.



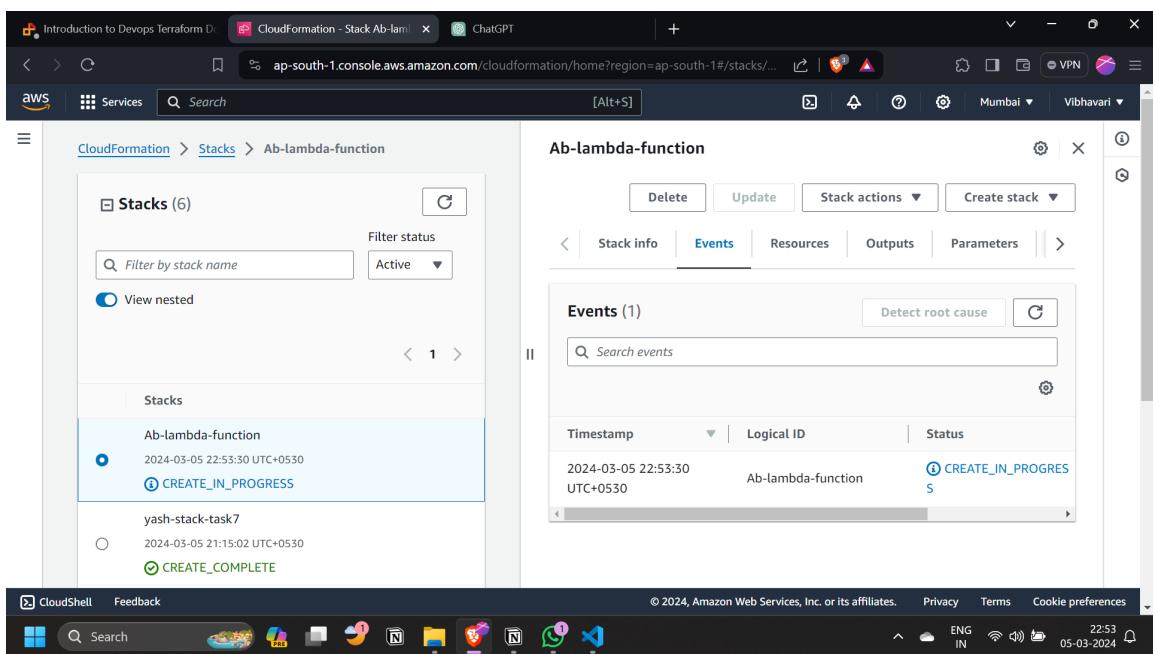
4. Specify Stack Details: Enter a stack name and click "Next".



5. Review and Create Stack: Review the stack details and click "Create stack" to initiate the creation process.



6. Wait for Stack Creation: Monitor the CloudFormation console until the stack status changes to "CREATE_COMPLETE".



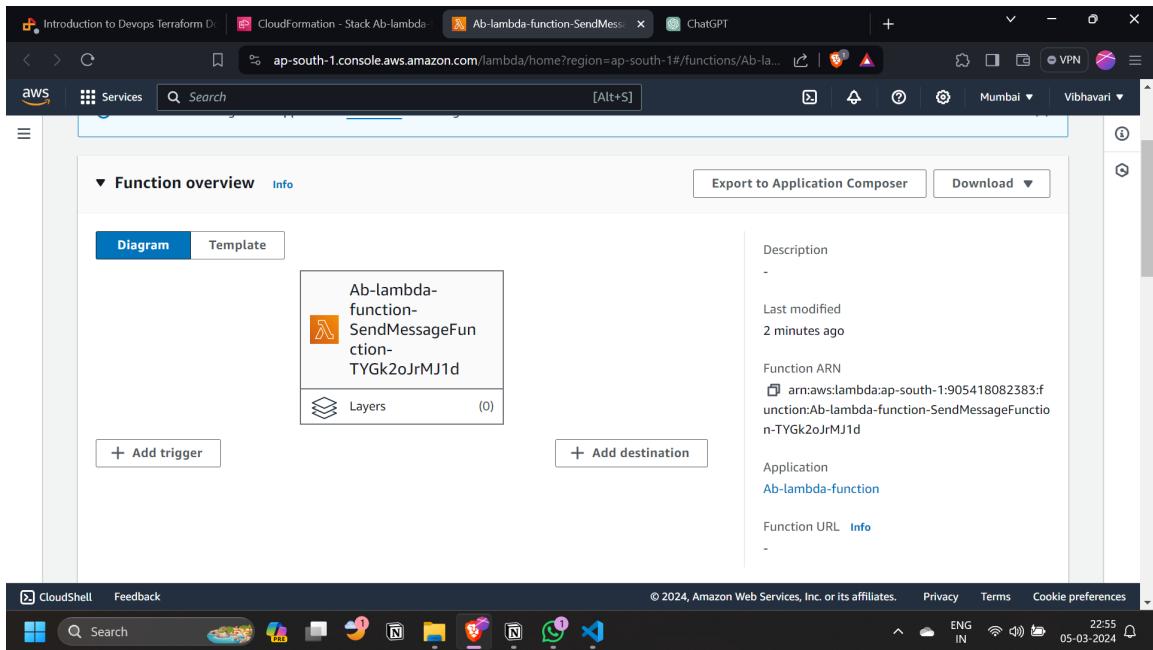
The screenshot shows the AWS CloudFormation console with the 'Events' tab selected for the 'Ab-lambda-function' stack. The table lists one event:

Timestamp	Logical ID	Status
2024-03-05 22:53:30 UTC+0530	Ab-lambda-function	CREATE_IN_PROGRESS

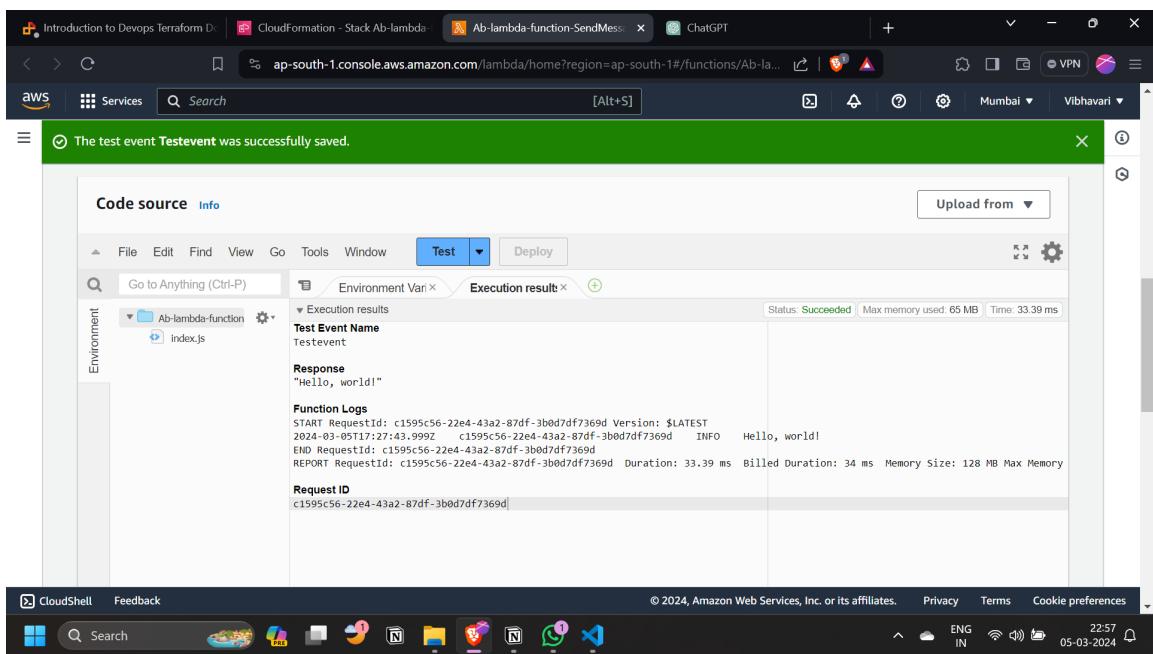
7. Verify Lambda Function: Confirm that the Lambda function is created by navigating to the Lambda service in the AWS Management Console.

The screenshot shows the AWS CloudFormation console with the 'Resources' tab selected for the 'Ab-lambda-function' stack. The table lists two resources:

Logical ID	Type
Ab-lambda-function-LambdaExecutionRole-WD1236KQdB03	AWS::IAM::Role
Ab-lambda-function-SendMessageFunction-TYGk2oJrMJ1d	AWS::Lambda::Function



8. Test Lambda Function: Test the Lambda function by configuring a test event with sample data and invoking the function manually from the Lambda console.



The Lambda function to store incoming data to DynamoDB and send it to an SQS queue using the CloudFormation template provided.

Implementation of Creating Lambda Function using Cloud formation template:

Sure, here are the important steps for implementing the CloudFormation template to create a Lambda function with DynamoDB and SQS access:

- 1. Prepare Your Template:** Modify the provided CloudFormation template by replacing `'YourDynamoDBTableName'` with your actual DynamoDB table name and `'YourSQSQueueURL'` with your actual SQS queue URL. Save the modified template with a `.yaml` extension.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template to create a Lambda function with DynamoDB and SQS access'

Resources:
  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Principal:
              Service: 'lambda.amazonaws.com'
              Action: 'sts:AssumeRole'
      Policies:
        - PolicyName: 'LambdaDynamoDBSQSPolicy'
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: 'Allow'
                Action:
                  - 'dynamodb:PutItem'
                Resource: '*'
              - Effect: 'Allow'
                Action:
                  - 'sns:SendMessage'
                Resource: 'arn:aws:sns:ap-south-1:905418'
```

```

082383:ab-queue'

LambdaFunction:
  Type: 'AWS::Lambda::Function'
Properties:
  Handler: 'index.lambda_handler'
  Role: !GetAtt LambdaExecutionRole.Arn
  Code:
    ZipFile: |
      import boto3
      import json
      import time

      dynamodb = boto3.client('dynamodb')
      sqs = boto3.client('sns')

      table_name = 'YourDynamoDBTableName'
      queue_url = 'YourSQSQueueURL'
      max_retries = 3

    def lambda_handler(event, context):
        try:
            data = json.loads(event['body'])

            response = dynamodb.put_item(
                TableName=table_name,
                Item={
                    'id': {'S': data['id']},
                    'payload': {'S': json.dumps(da
ta)}
                }
            )

            sqs.send_message(
                QueueUrl=queue_url,
                MessageBody=json.dumps(data)
            )

```

```

        return {
            'statusCode': 200,
            'body': json.dumps('Data stored in
DynamoDB and sent to SQS successfully')
        }
    except Exception as e:
        retries = 0
        while retries < max_retries:
            retries += 1
            time.sleep(retries * 2)
        try:
            response = lambda_handler(event,
t, context)
            return response
        except Exception as e:
            if retries == max_retries:
                return {
                    'statusCode': 500,
                    'body': json.dumps('Fa
iled to store data in DynamoDB and send to SQS after max
imum retries')
                }
            else:
                continue

```

Runtime: python3.8

Timeout: 30

LambdaPermissionDynamoDB:

Type: 'AWS::Lambda::Permission'

Properties:

Action: 'lambda:InvokeFunction'

FunctionName: !GetAtt LambdaFunction.Arn

Principal: 'dynamodb.amazonaws.com'

LambdaPermissionSQS:

Type: 'AWS::Lambda::Permission'

Properties:

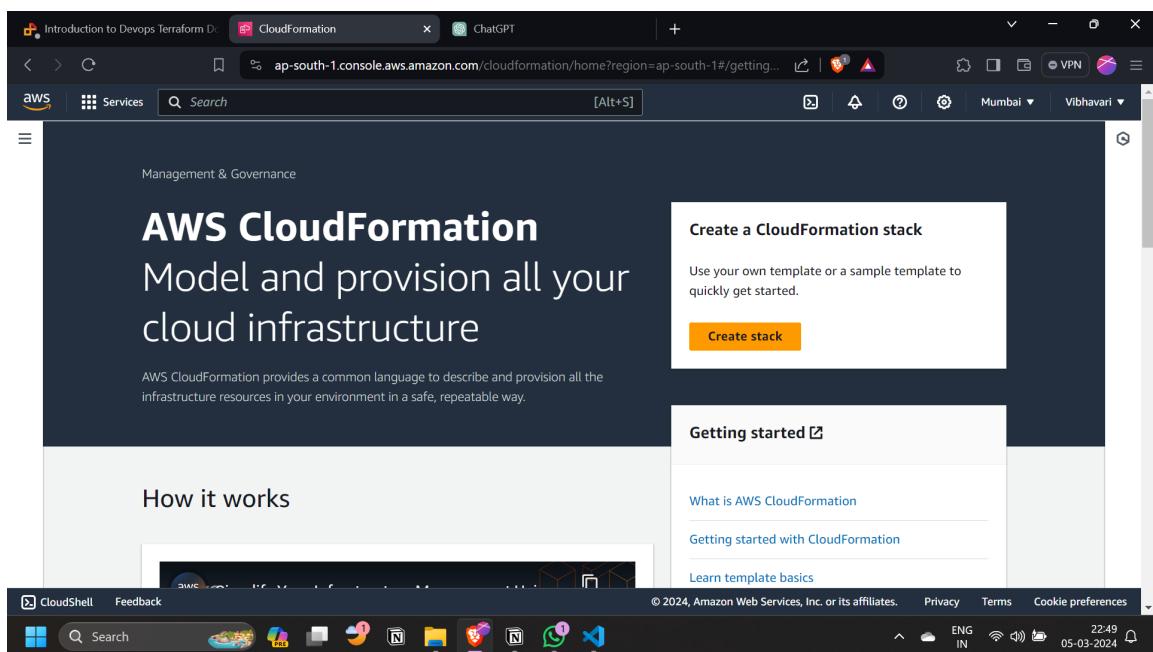
```
Action: 'lambda:InvokeFunction'  
FunctionName: !GetAtt LambdaFunction.Arn  
Principal: 'sns.amazonaws.com'
```

Outputs:

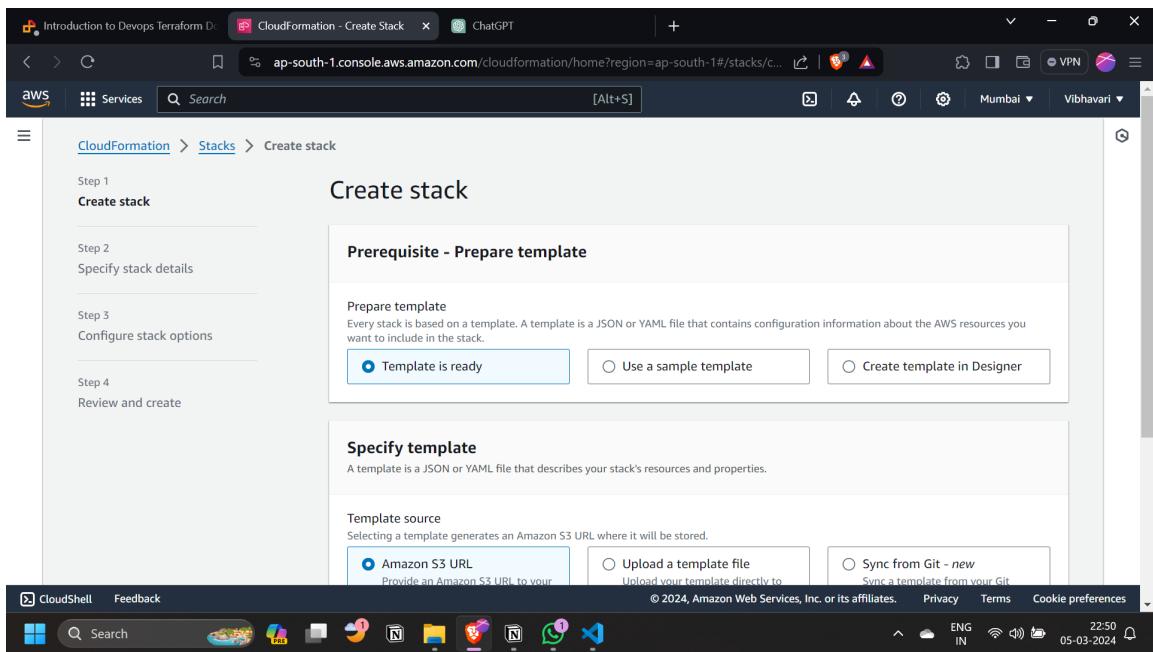
LambdaFunctionARN:

```
Description: 'ARN of the created Lambda function'  
Value: !GetAtt LambdaFunction.Arn
```

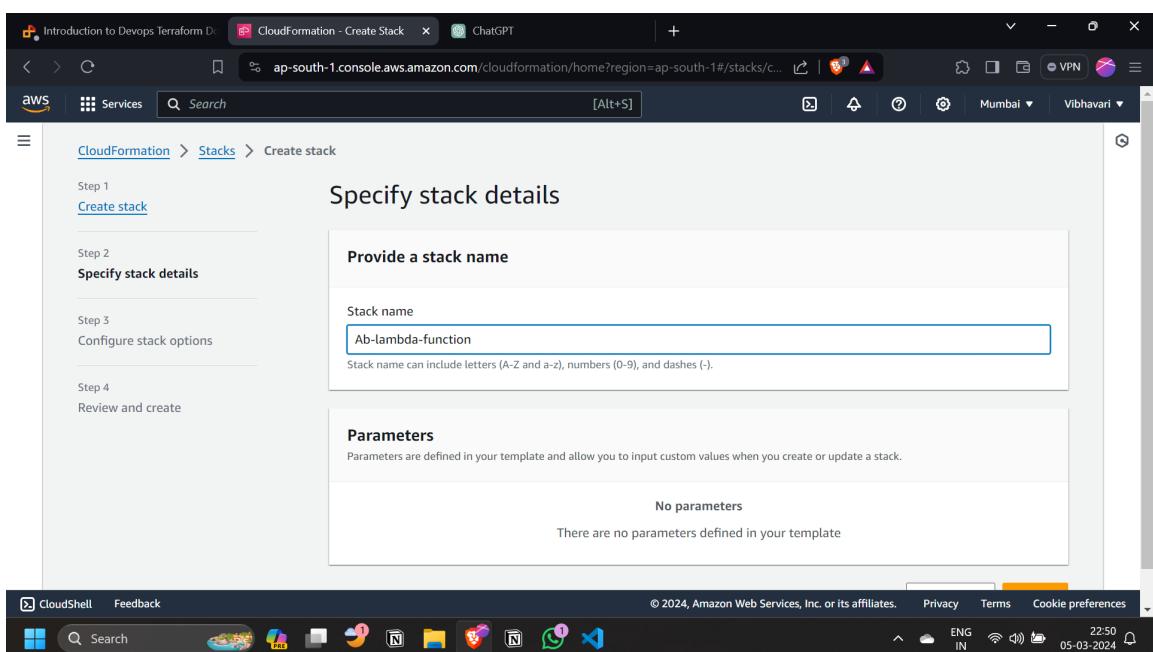
2. **Navigate to CloudFormation:** Go to the AWS Management Console and navigate to the CloudFormation service.



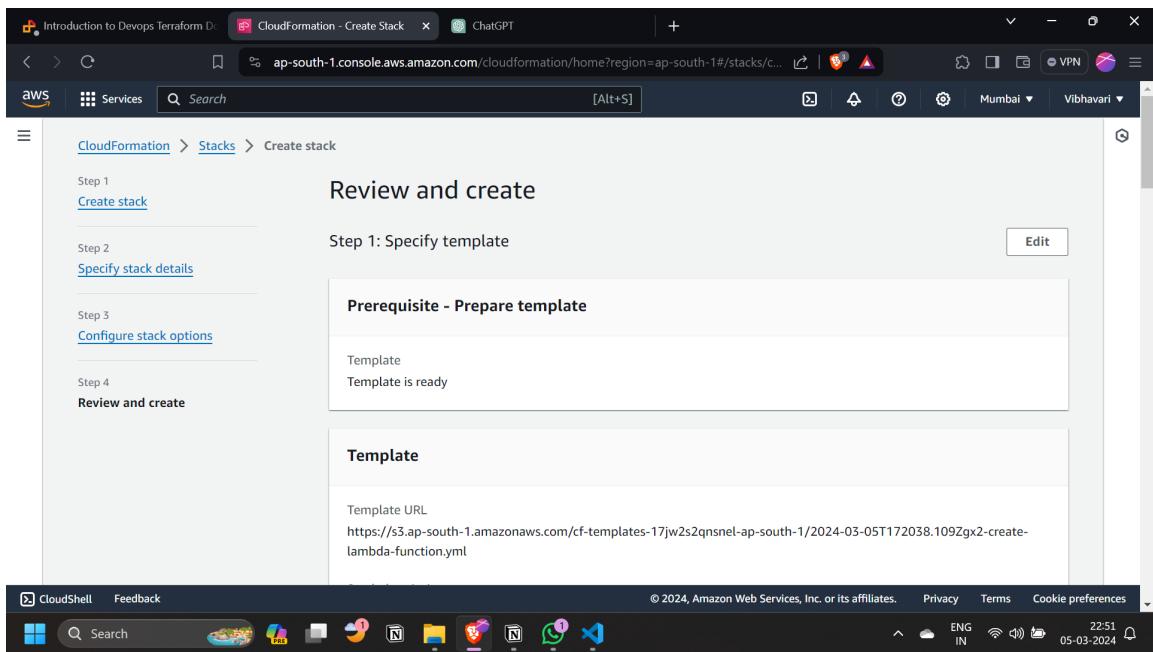
3. **Create a New Stack:** Click on "Create stack" and upload the modified CloudFormation template file.



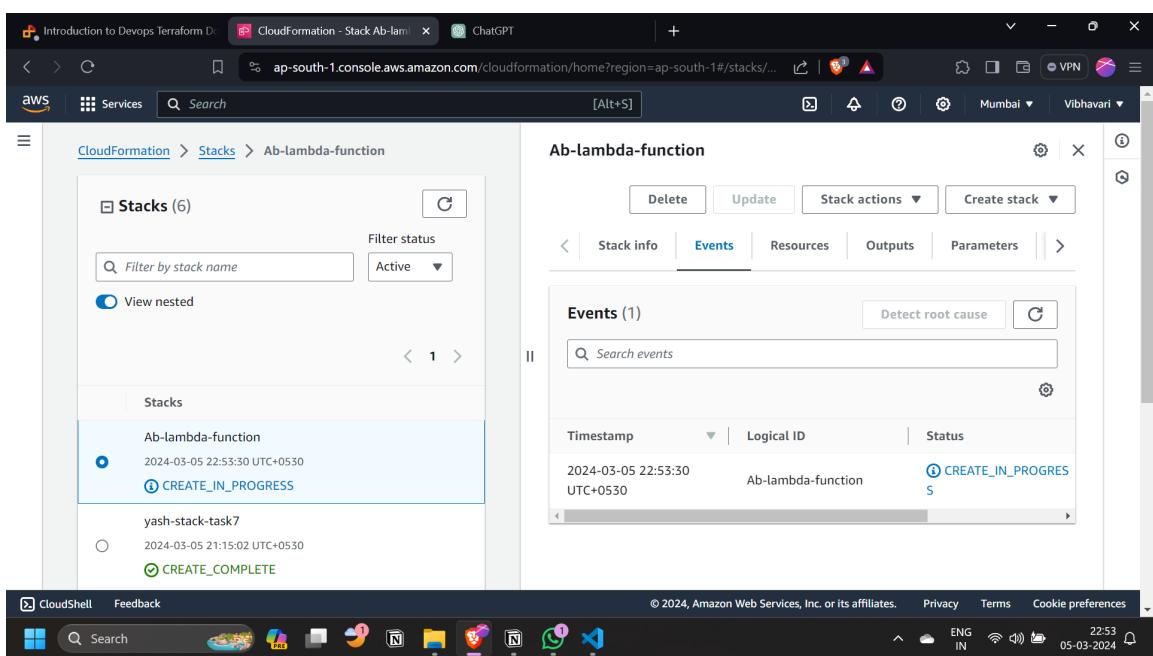
4. Specify Stack Details: Enter a stack name and click "Next".



5. Review and Create Stack: Review the stack details and click "Create stack" to initiate the creation process.



6. Wait for Stack Creation: Monitor the CloudFormation console until the stack status changes to "CREATE_COMPLETE".



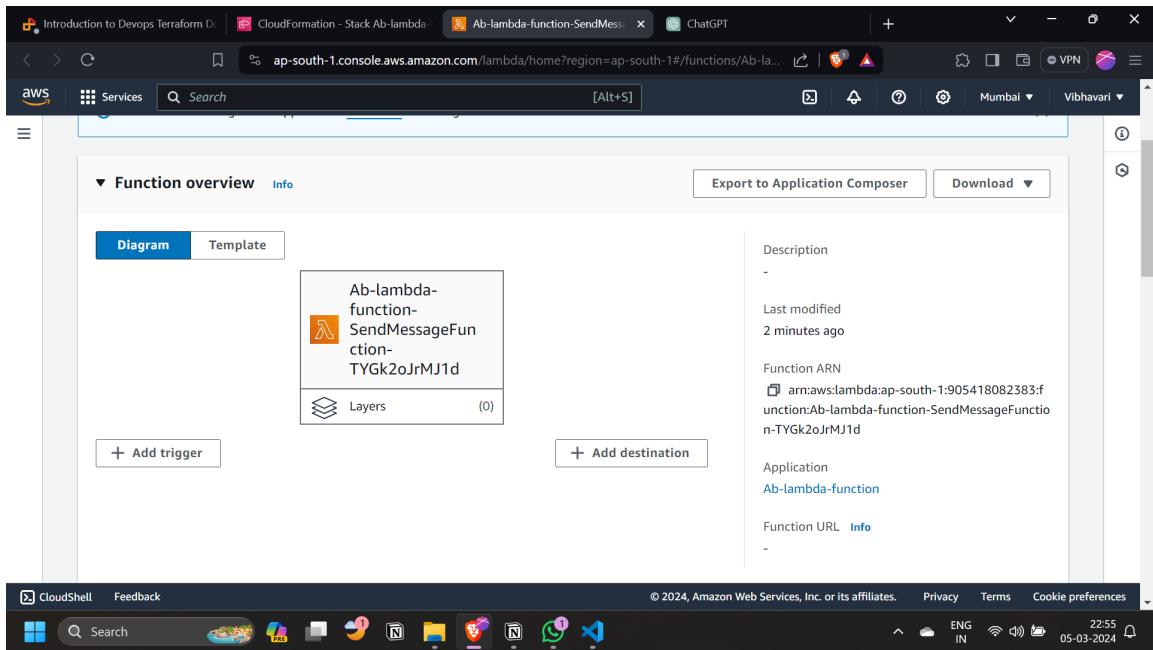
The screenshot shows the AWS CloudFormation console with the 'Events' tab selected for the 'Ab-lambda-function' stack. The table displays one event entry:

Timestamp	Logical ID	Status
2024-03-05 22:53:30 UTC+0530	Ab-lambda-function	CREATE_IN_PROGRESS

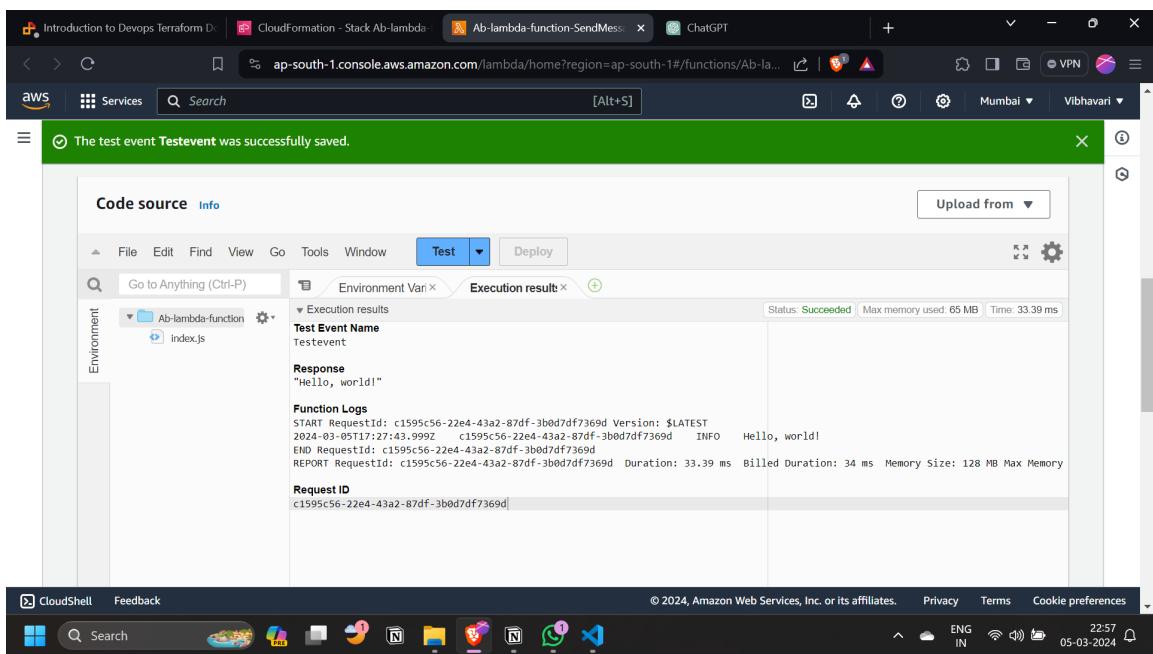
7. Verify Lambda Function: Confirm that the Lambda function is created by navigating to the Lambda service in the AWS Management Console.

The screenshot shows the AWS CloudFormation console with the 'Resources' tab selected for the 'Ab-lambda-function' stack. The table displays two resources:

Logical ID	Type
Ab-lambda-function-LambdaExecutionRole-WD1236KQdB03	AWS::IAM::Role
Ab-lambda-function-SendMessageFunction-TYGk2oJrMJ1d	AWS::Lambda::Function



8. Test Lambda Function: Test the Lambda function by configuring a test event with sample data and invoking the function manually from the Lambda console.



The Lambda function to store incoming data to DynamoDB and send it to an SQS queue using the CloudFormation template provided.

Task 8: Attach Lambda to SQS queue

To attach a Lambda function to an SQS queue, you can use an SQS trigger configuration in the Lambda function settings. This configuration ensures that the Lambda function is invoked whenever a message is available in the SQS queue.

Here's how you can achieve SQS integration with Lambda and store data to a DynamoDB table when data comes to the SQS queue:

1. Lambda Function:

First, ensure that you have a Lambda function created. If not, create a Lambda function that will process messages from the SQS queue and store data to the DynamoDB table as per your requirements.

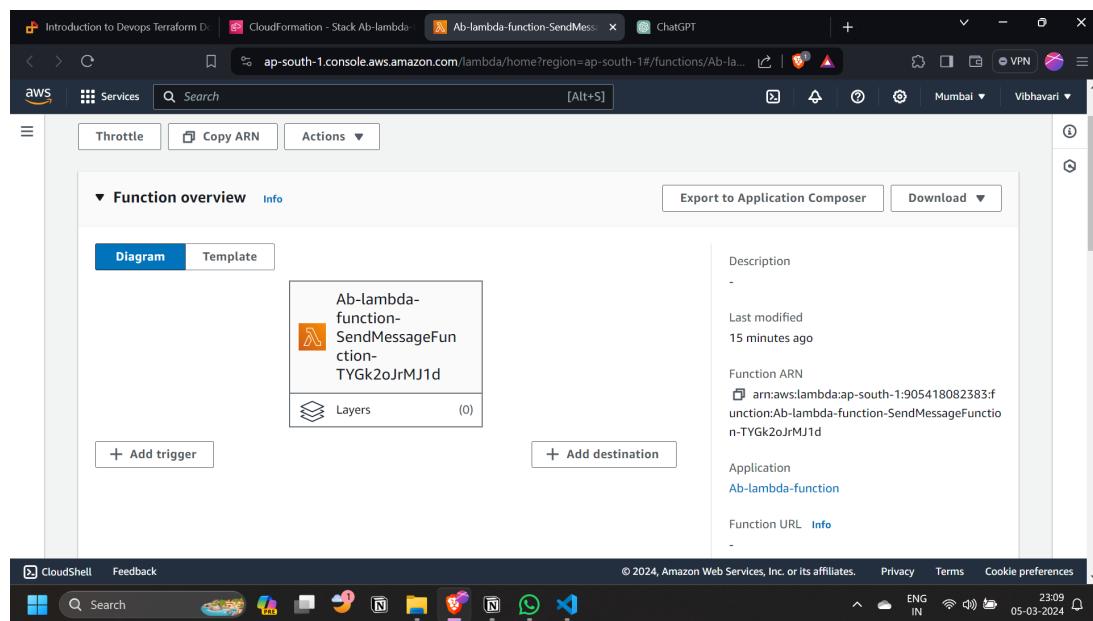
2. SQS Queue:

Make sure you have an SQS queue created where data will be sent. If not, create an SQS queue in the AWS Management Console.

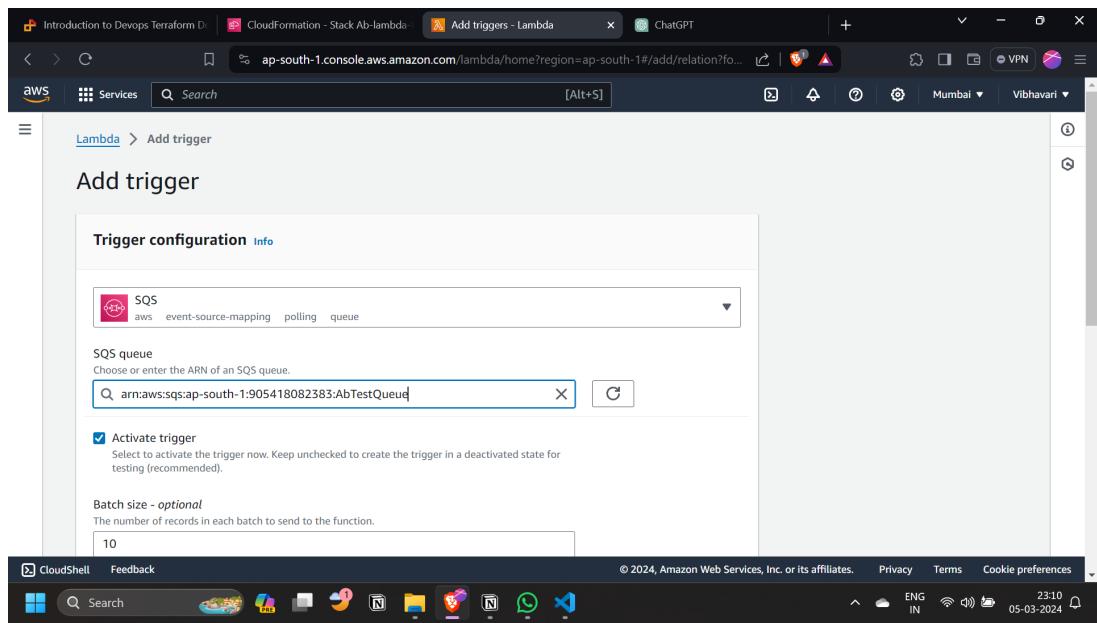
3. Configure Lambda-SQS Integration:

Go to the Lambda service in the AWS Management Console. Find the Lambda function you want to attach to the SQS queue. Then, follow these steps:

- Open the Lambda function configuration.



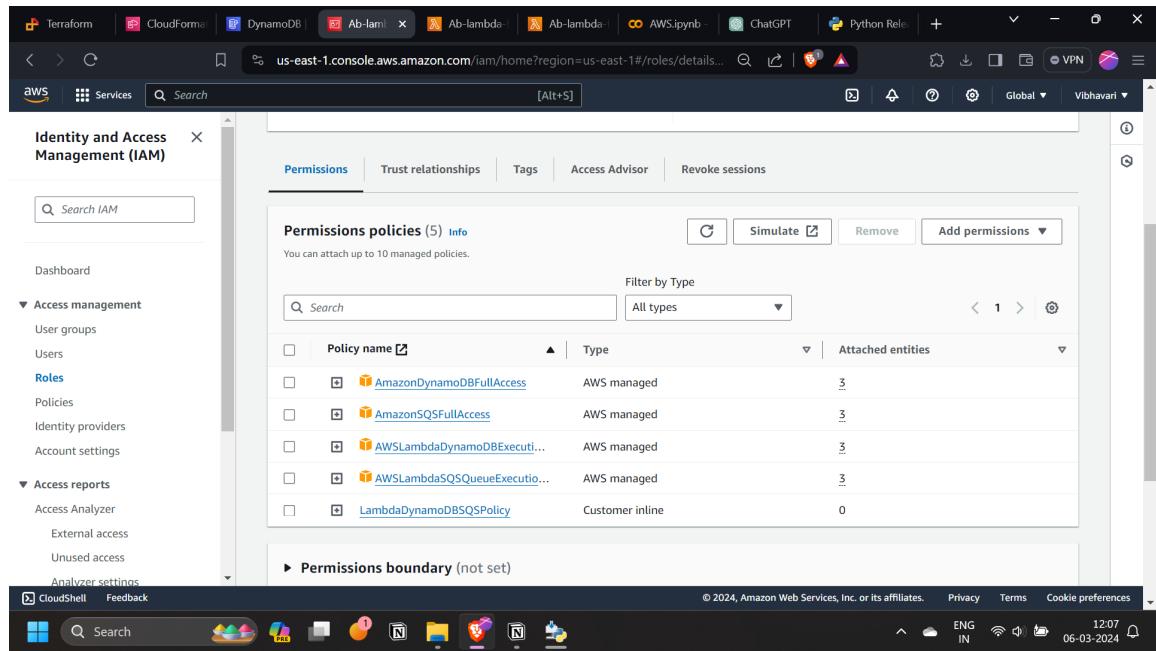
- Click on "Add trigger".



4. Ensure Lambda Permissions:

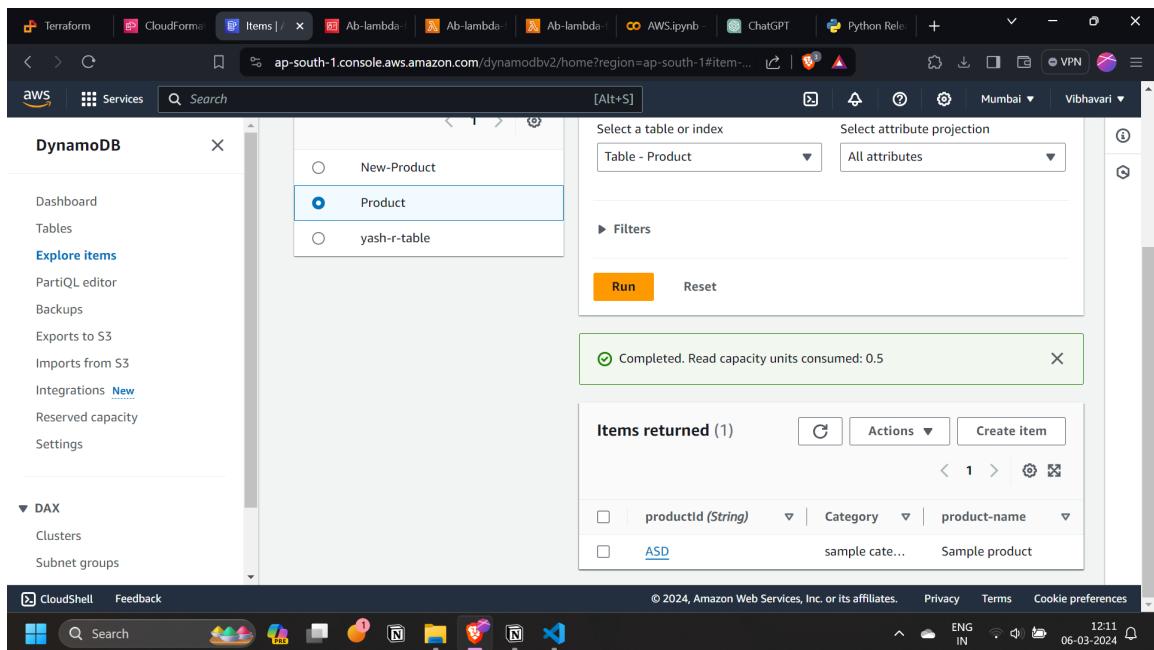
Ensure that the Lambda function's execution role has the necessary permissions to receive messages from the SQS queue. You might need to add the

`SQS:ReceiveMessage` permission to the Lambda execution role



5. Testing:

Test the integration by sending messages to the SQS queue. Monitor the Lambda function's execution to verify that it processes the messages and stores the data in the DynamoDB table correctly.



Python code to send data to SQS

```
import boto3
import json

# Create an SQS client
region = 'ap-south-1'
# Replace 'your_access_key_id' and 'your_secret_access_key'
# with your AWS credentials
aws_access_key_id = 'AKIAYS2NU4DZW5NYCCFS'
aws_secret_access_key = 'ZUwKcoxwbEtM2yqXVQhXmp4ce0muVvztGv
NoLLri'

# URL of your SQS queue
queue_url = '<https://sns.ap-south-1.amazonaws.com/90541808
2383/ab-queue>'

sns = boto3.client('sns', region_name=region, aws_access_ke
y_id=aws_access_key_id, aws_secret_access_key=aws_secret_ac
cess_key)

# Data to send
data = {
    'ProductID': '123467',
```

```
'ProductName': 'Sample Product',
'Quantity': 200,
'Price': 50,
'Discount': 5,
'Category': 'Sample Category',
'RemainingStock': 100
}

# Send data to SQS
response = sqs.send_message(
    QueueUrl=queue_url,
    MessageBody=json.dumps(data)
)

print("Message sent to SQS:", response['MessageId'])
```

By following these steps, you can attach a Lambda function to an SQS queue and ensure that data sent to the SQS queue is processed and stored in the DynamoDB table by the Lambda function.